Pengantar Pohon (Bagian 2)

Pohon Perentang (*Spanning Tree*), Traversal pada Pohon (*Tree Traversal-Suplemen*), dan Beberapa Aplikasi Pohon (Suplemen)

MZI

Fakultas Informatika
Telkom University

FIF Tel-U

Mei 2023

Acknowledgements

Slide ini disusun berdasarkan materi yang terdapat pada sumber-sumber berikut:

1. *Discrete Mathematics and Its Applications*, Edisi 8, 2019, oleh K. H. Rosen (acuan utama).
2. *Discrete Mathematics with Applications*, Edisi 5, 2018, oleh S. S. Epp.
3. *Mathematics for Computer Science*. MIT, 2010, oleh E. Lehman, F. T. Leighton, A. R. Meyer.
4. *Slide* kuliah Matematika Diskret 2 (2012) di Fasilkom UI oleh B. H. Widjaja.
5. Slide kuliah Matematika Diskret 2 di Fasilkom UI oleh Tim Dosen.
6. *Slide* kuliah Matematika Diskrit di Telkom University oleh B. Purnama dan rekan-rekan.

Beberapa gambar dapat diambil dari sumber-sumber di atas. *Slide* ini ditujukan untuk keperluan akademis di lingkungan FIF Telkom University. Jika Anda memiliki saran/ pendapat/ pertanyaan terkait materi dalam *slide* ini, silakan kirim email ke `<pleasedontspam>@telkomuniversity.ac.id`.

Bahasan

1. Pohon Perentang (Spanning Tree)
2. Traversal pada Pohon (Tree Traversal - Suplemen)
3. Aplikasi Pohon – Pohon Urai (Parse Tree - Suplemen)
4. Aplikasi Pohon – Pohon Keputusan (Decision Tree - Suplemen)
5. Aplikasi Pohon – Notasi Infiks, Prefiks, dan Postfiks (Suplemen)

Bahasan

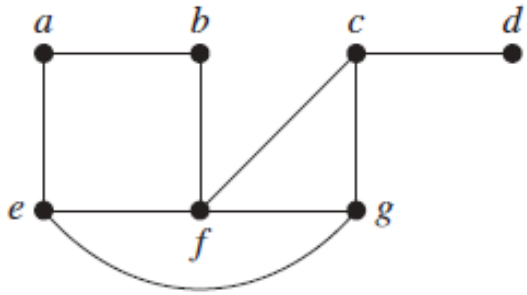
Pohon Perentang (*Spanning Tree*)

Pohon (*Spanning Tree*)

Pohon perentang dari suatu graf G adalah subgraf peretang dari G yang berupa pohon

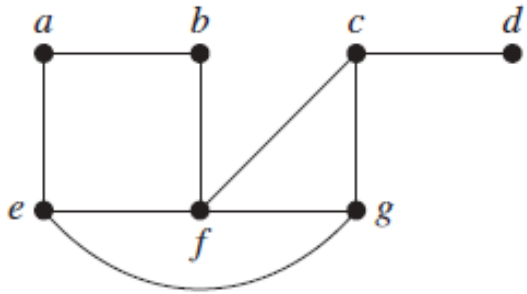
Hal ini berarti $T = (V_T, E_T)$ adalah pohon perentang dari $G = (V_G, E_G)$ bila T adalah pohon dan $V_T = V_G$. Pohon perentang dari suatu graf dapat diperoleh dengan memutus sirkuit sederhana yang terdapat di graf tersebut.

Graf asal:

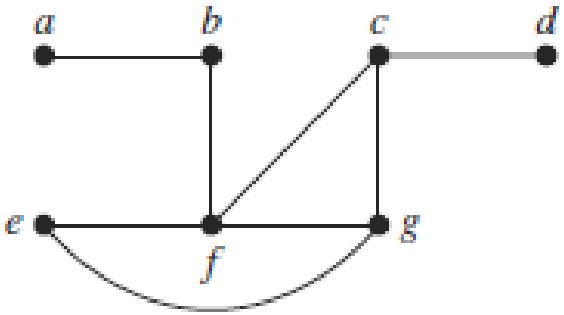


Konstruksi pohon:

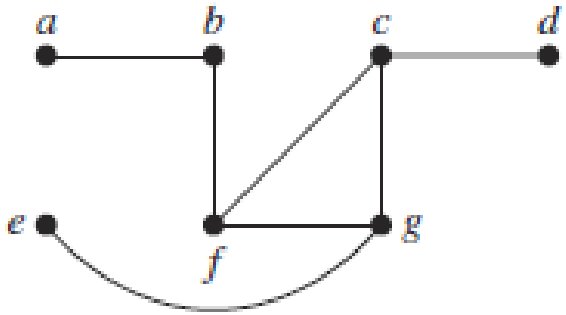
Graf asal:



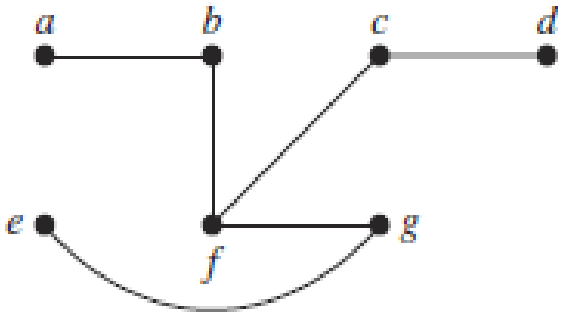
Konstruksi pohon:



Edge removed: $\{a, e\}$

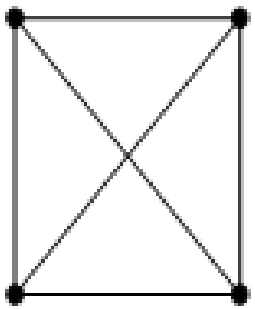


$\{e, f\}$

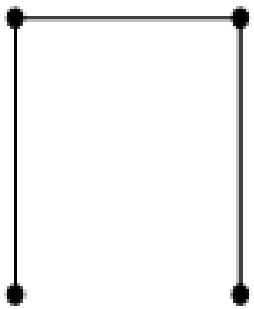


$\{c, g\}$

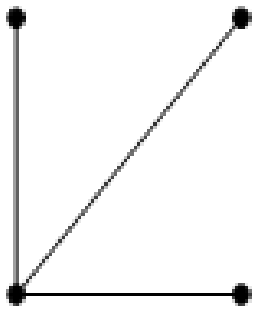
Sebuah graf dapat memiliki lebih dari satu pohon perentang. Berikut adalah beberapa pohon perentang dari K_4.



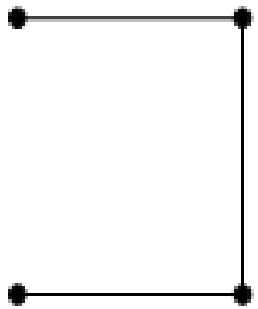
G



T_1



T_2



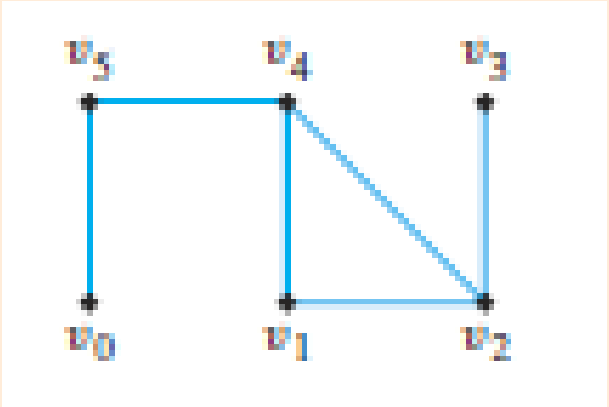
T_3

T_4

Latihan 1: Mencari Semua Pohon Perentang

Latihan

Carilah semua pohon perentang dari graf berikut.

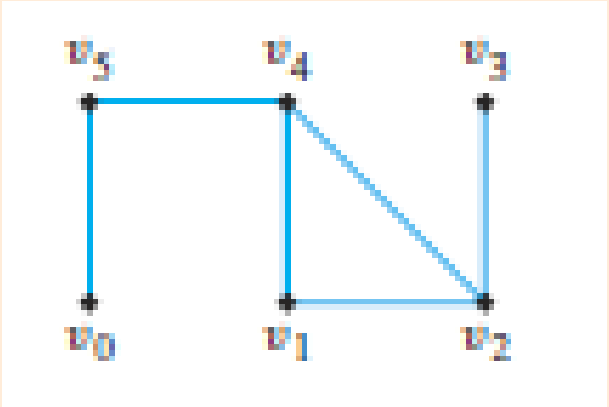


Solusi: Kita memiliki:

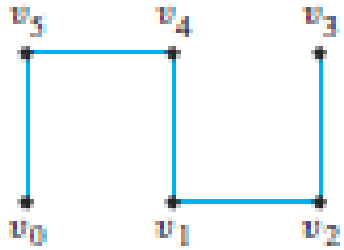
Latihan 1: Mencari Semua Pohon Perentang

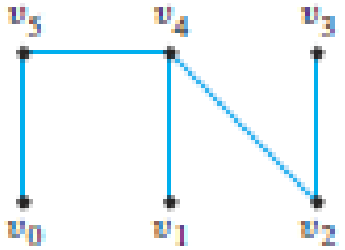
Latihan

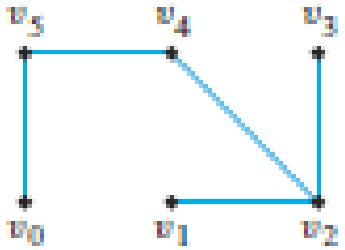
Carilah semua pohon perentang dari graf berikut.



Solusi: Kita memiliki:







Sifat-sifat Pohon Perentang dari Suatu Graf

- Setiap graf terhubung setidaknya memiliki satu buah pohon perentang.
- Graf yang tak terhubung dengan k komponen setidaknya memiliki k komponen pohon perentang yang disebut sebagai hutan perentang (*spanning forest*).

Pohon Perentang Minimum (*Minimum Spanning Tree*)

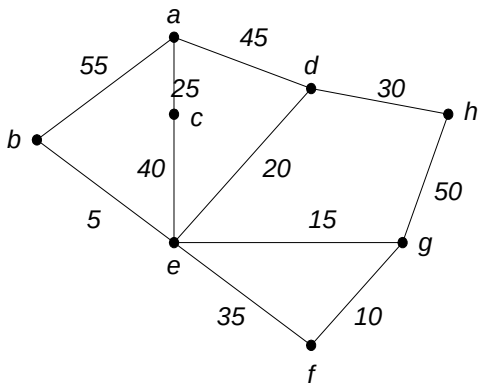
Pohon Perentang Minimum (*Minimum Spanning Tree*)

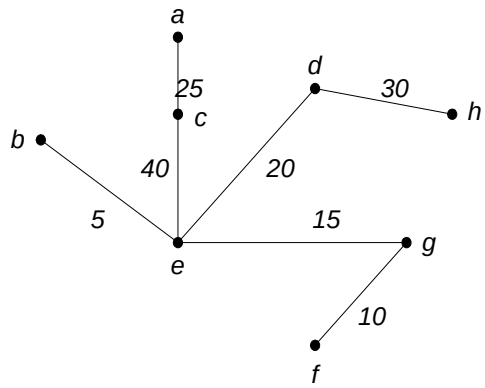
Sebuah graf berbobot yang terhubung mungkin memiliki lebih dari satu pohon perentang. Pohon perentang dengan bobot minimum selanjutnya dinamakan sebagai pohon perentang minimum (*minimum spanning tree*).

Pohon Perentang Minimum (*Minimum Spanning Tree*)

Pohon Perentang Minimum (*Minimum Spanning Tree*)

Sebuah graf berbobot yang terhubung mungkin memiliki lebih dari satu pohon perentang. Pohon perentang dengan bobot minimum selanjutnya dinamakan sebagai pohon perentang minimum (*minimum spanning tree*).





Untuk menentukan pohon perentang minimum dari suatu graf, kita dapat memakai dua algoritma, yaitu algoritma Prim dan algoritma Kruskal.

Algoritma Prim untuk Pohon Perentang Minimum

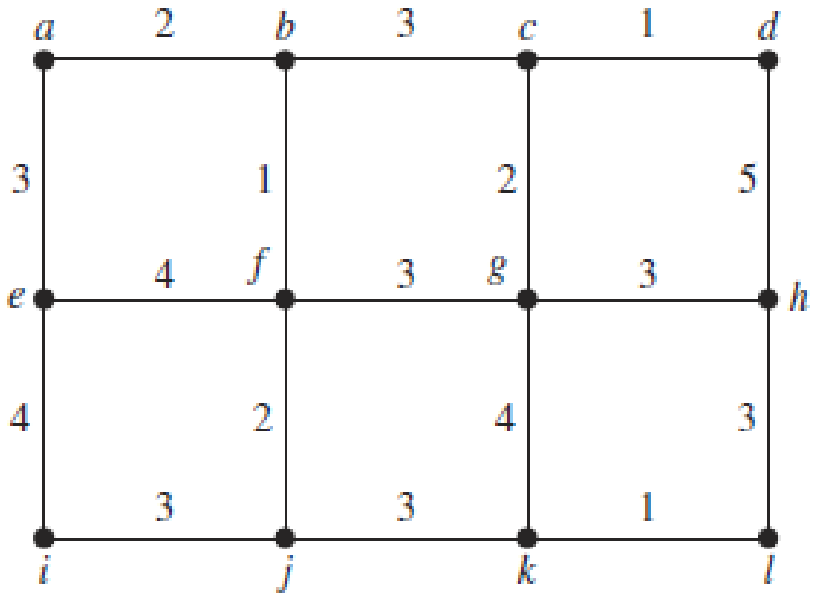
Algoritma Prim

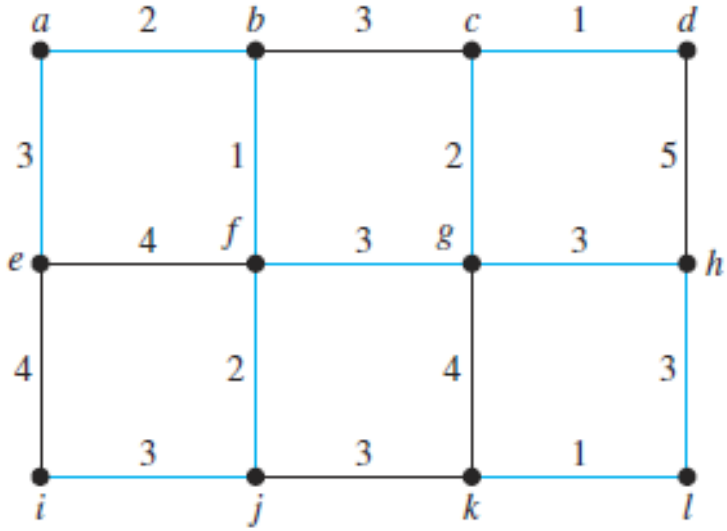
1. `Masukan:` sebuah graf $G = (V_G, E_G)$ yang terhubung dan berbobot, $|V_G| = n$.
2. `Inisialisasi:` $T = (V, E)$ memuat **semua simpul** pada G dan $E = \{e\}$ dengan e berbobot **minimum**
3. `for` $i := 1$ `to` $n - 2$
4. Pilih $e = \{u, v\}$ sebagai sisi yang memenuhi **semua kriteria** berikut:
5. berbobot minimum `and` bertumpuan pada sebuah simpul di T
6. yang bersisian dengan suatu sisi
7. `if` $T' := (V, E \cup \{e\})$ tidak memuat sirkuit sederhana
8. $T := (V, E \cup \{e\})$
9. `else`
10. $T := (V, E)$
11. `Keluaran:` $T = (V, E)$ adalah pohon perentang minimum.

Algoritma Prim adalah salah satu contoh algoritma *greedy*, yaitu algoritma yang selalu mengambil pilihan terbaik (sisi berbobot terkecil) pada setiap iterasinya.

Ilustrasi Algoritma Prim

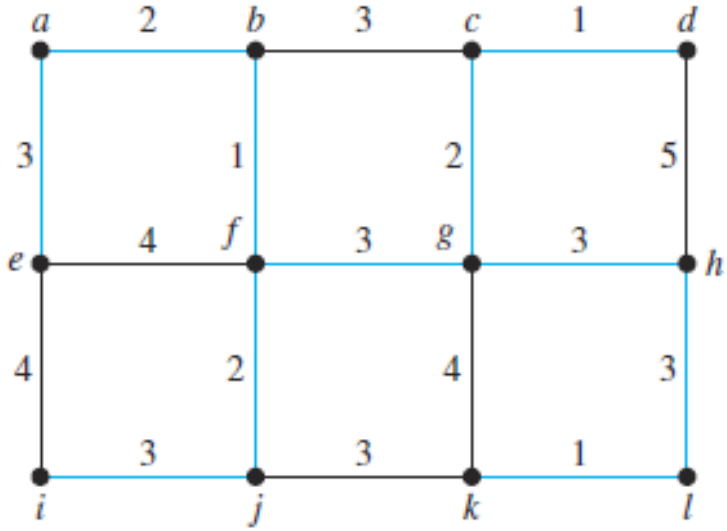
Misakan G adalah graf berikut.





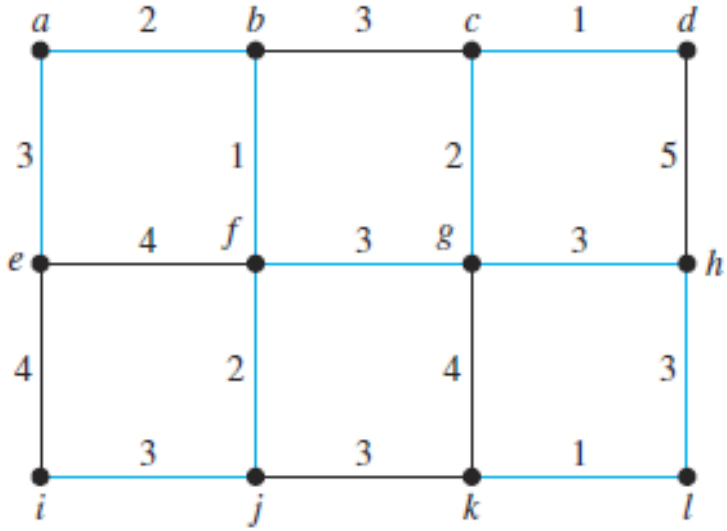
Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi						



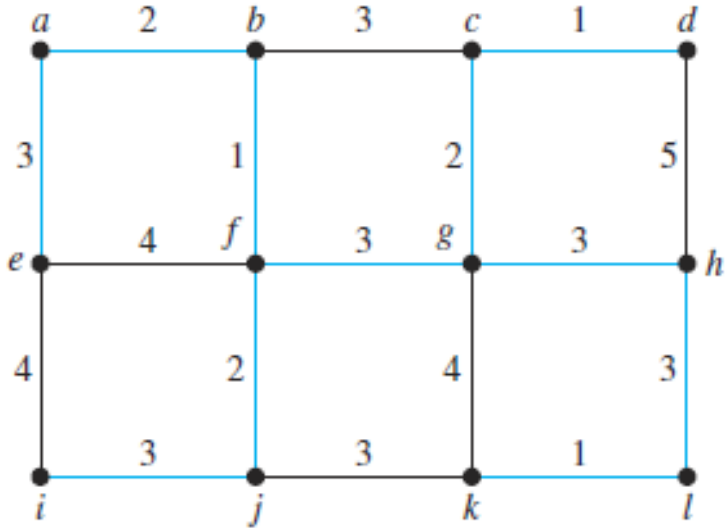
Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$					



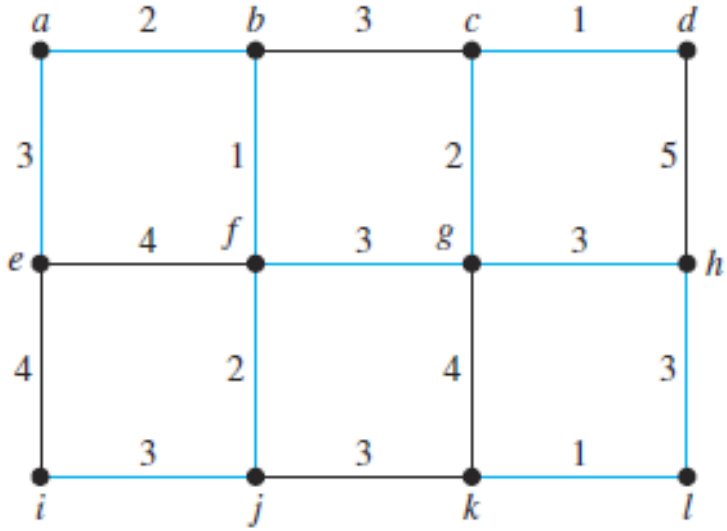
Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$				



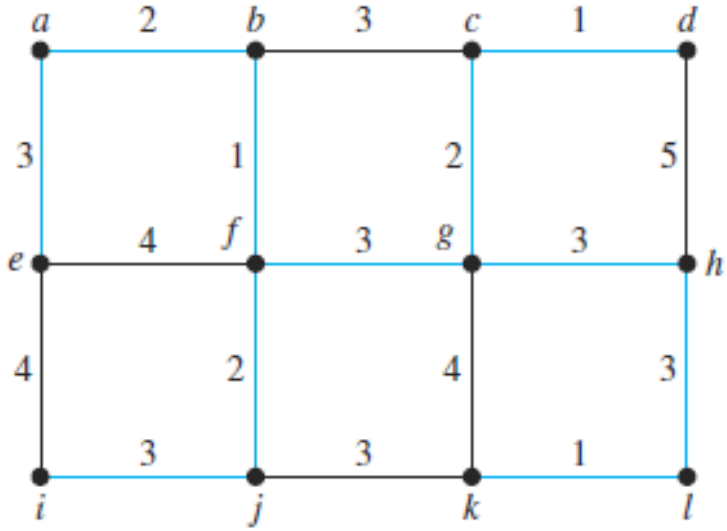
Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$			



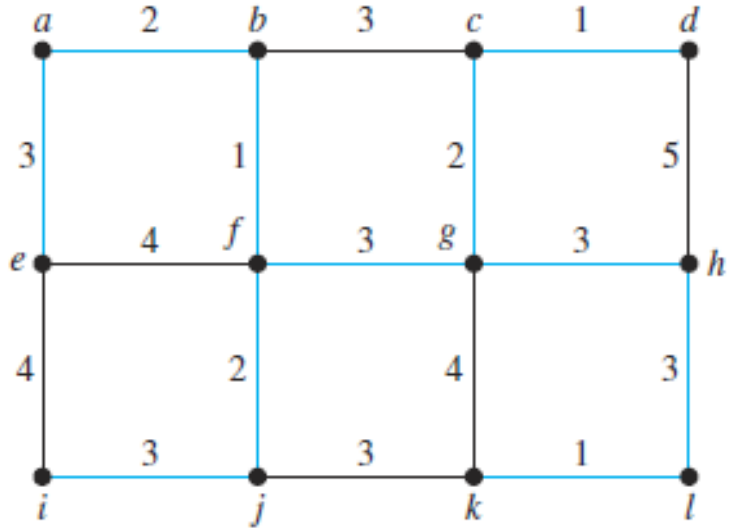
Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$		



Algoritma Prim bekerja sebagai berikut:

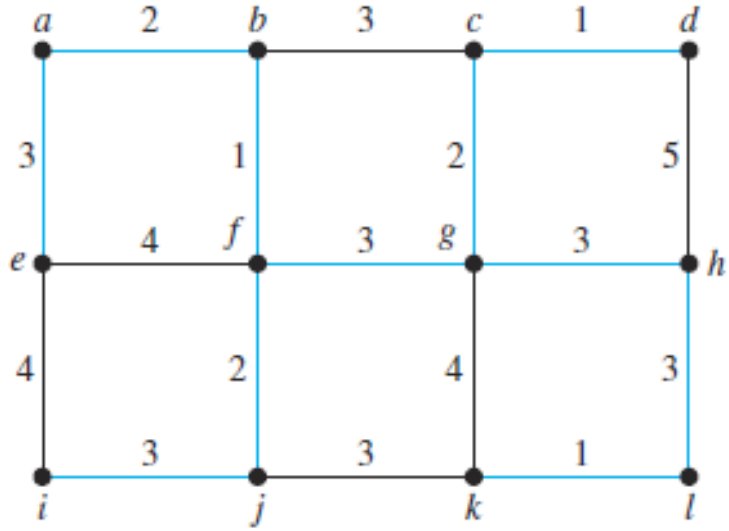
Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	



Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Bobot	1	2	2	3	3	3

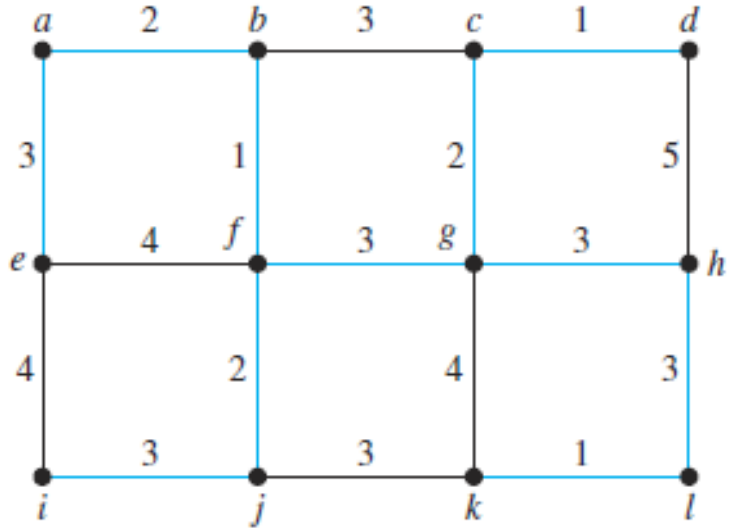
Pilihan ke-	7	8	9	10	11	Total
Sisi						



Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Bobot	1	2	2	3	3	3

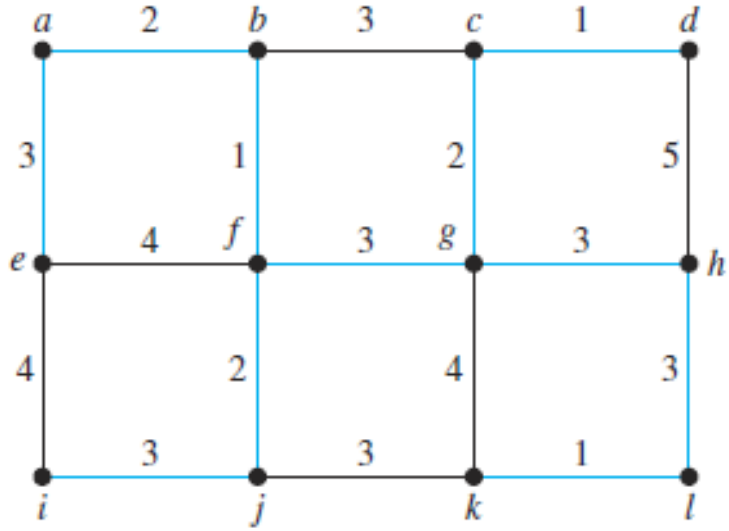
Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{g, c\}$					



Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Bobot	1	2	2	3	3	3

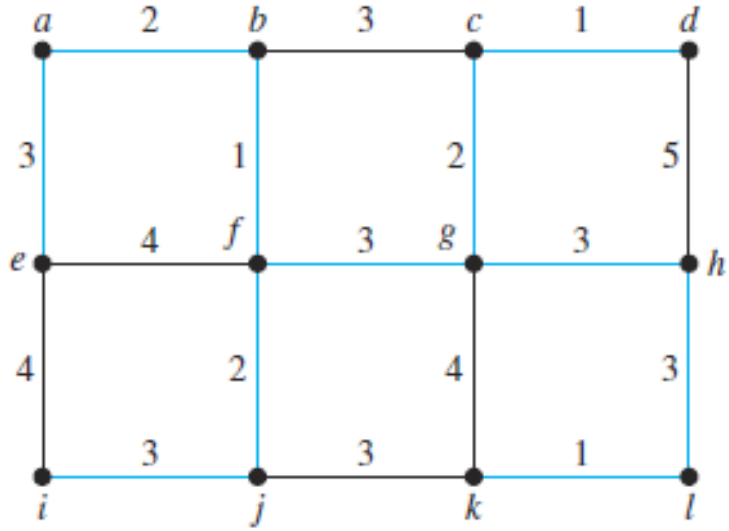
Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{g, c\}$	$\{c, d\}$				



Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Bobot	1	2	2	3	3	3

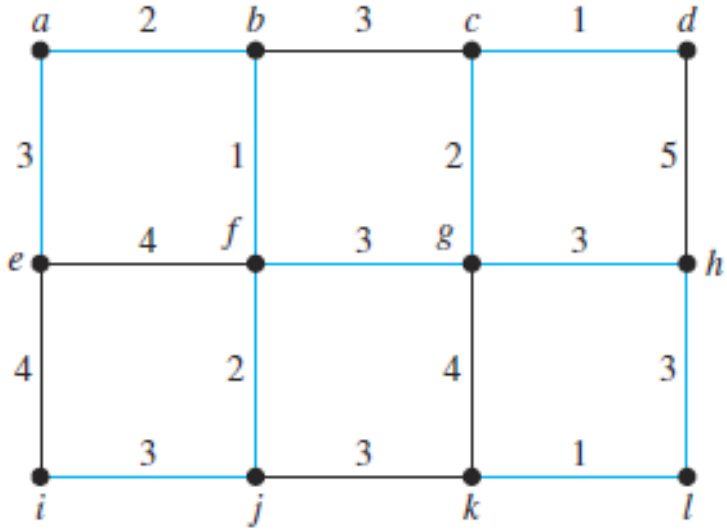
Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{g, c\}$	$\{c, d\}$	$\{g, h\}$			



Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Bobot	1	2	2	3	3	3

Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{g, c\}$	$\{c, d\}$	$\{g, h\}$	$\{h, l\}$		



Algoritma Prim bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Bobot	1	2	2	3	3	3

Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{g, c\}$	$\{c, d\}$	$\{g, h\}$	$\{h, l\}$	$\{l, k\}$	
Bobot	2	1	3	3	1	24

Algoritma Kruskal untuk Pohon Perentang Minimum

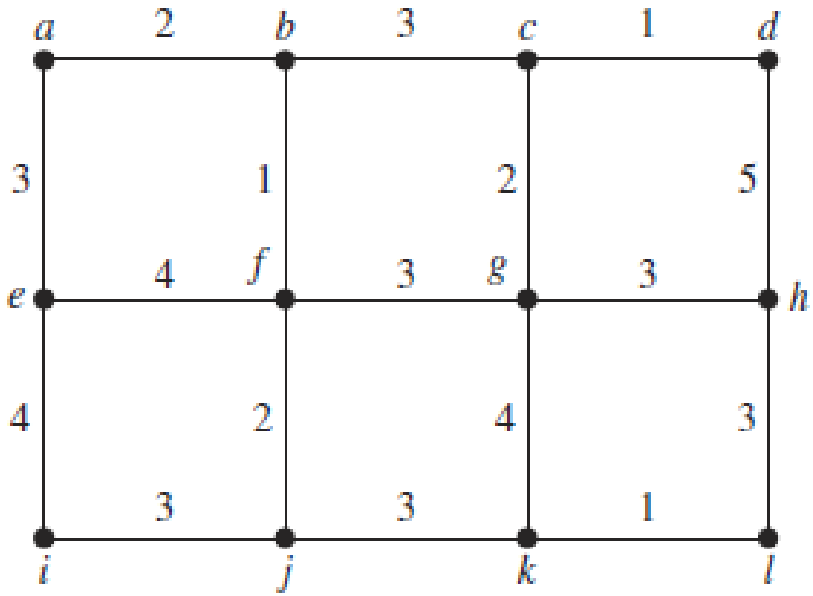
Algoritma Kruskal

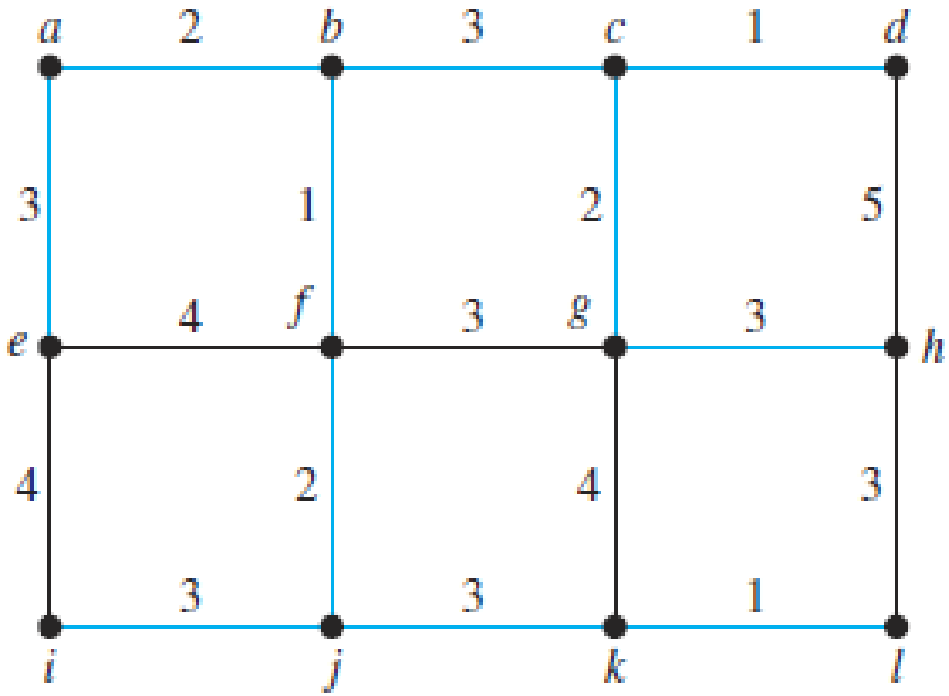
1. `Masukan:` sebuah graf $G = (V_G, E_G)$ yang terhubung dan berbobot, $|V_G| = n$.
2. `Inisialisasi:`
3. $\quad T = (V, E)$ dengan $V = \emptyset$ dan $E = \emptyset$.
4. $\quad$ Lakukan pengurutan (*sorting*) terhadap himpunan sisi E_G
5. `for` $i := 1$ `to` $n - 1$
6. $\quad$ Pilih $e = \{u, v\}$ dari E_G yang telah terurut
7. $\quad$ `if` $T' = (V, E \cup \{e\})$ tidak memuat sirkuit sederhana
8. $\quad\quad T = (V, E \cup \{e\})$
9. $\quad$ `else`
10. $\quad\quad T := (V, E)$
11. `Keluaran:` $T = (V, E)$ adalah pohon perentang minimum.

Algoritma Kruskal adalah salah satu contoh algoritma *greedy*, yaitu algoritma yang selalu mengambil pilihan terbaik (sisi berbobot terkecil) pada setiap langkah iterasinya.

Ilustrasi Algoritma Kruskal

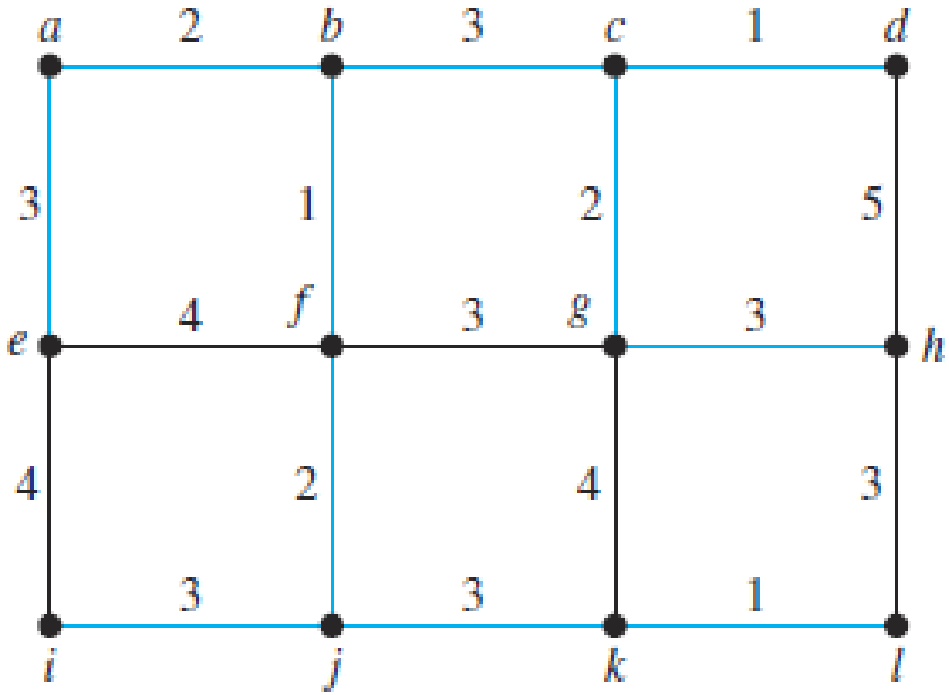
Misakan G adalah graf berikut.





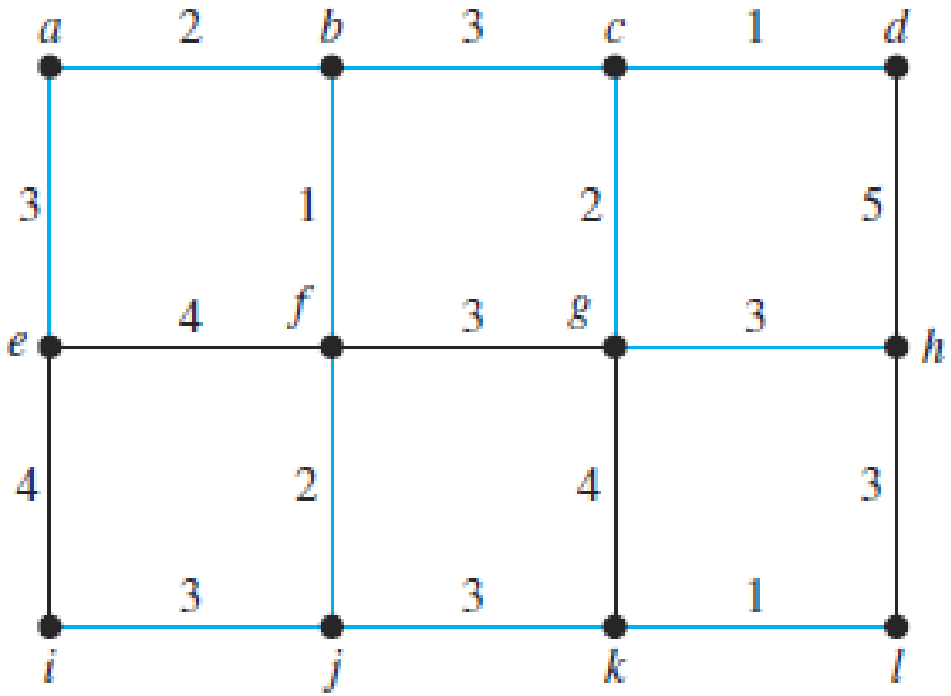
Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi						



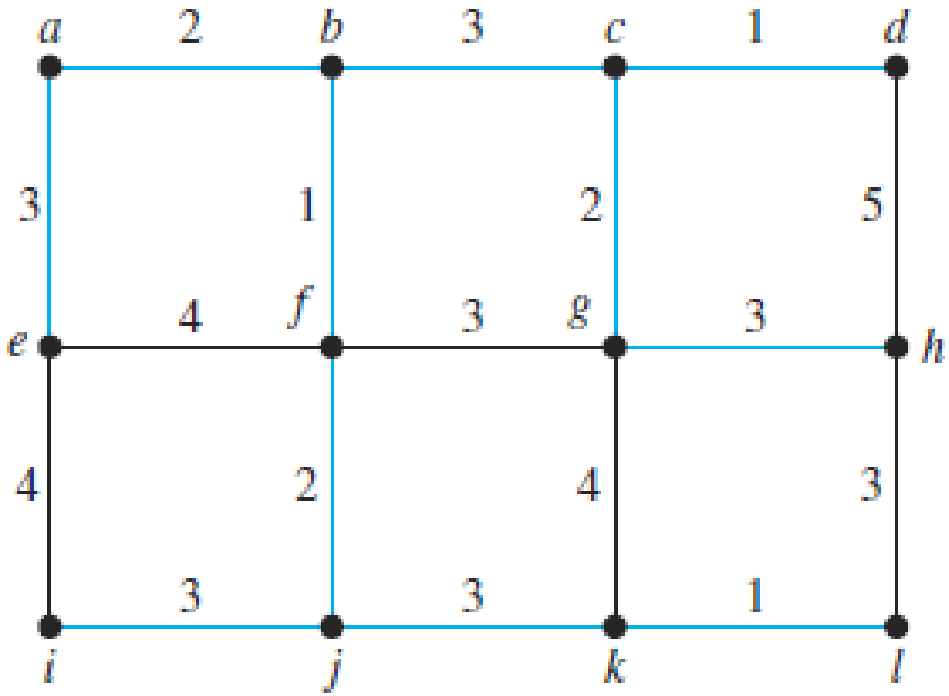
Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c, d\}$					



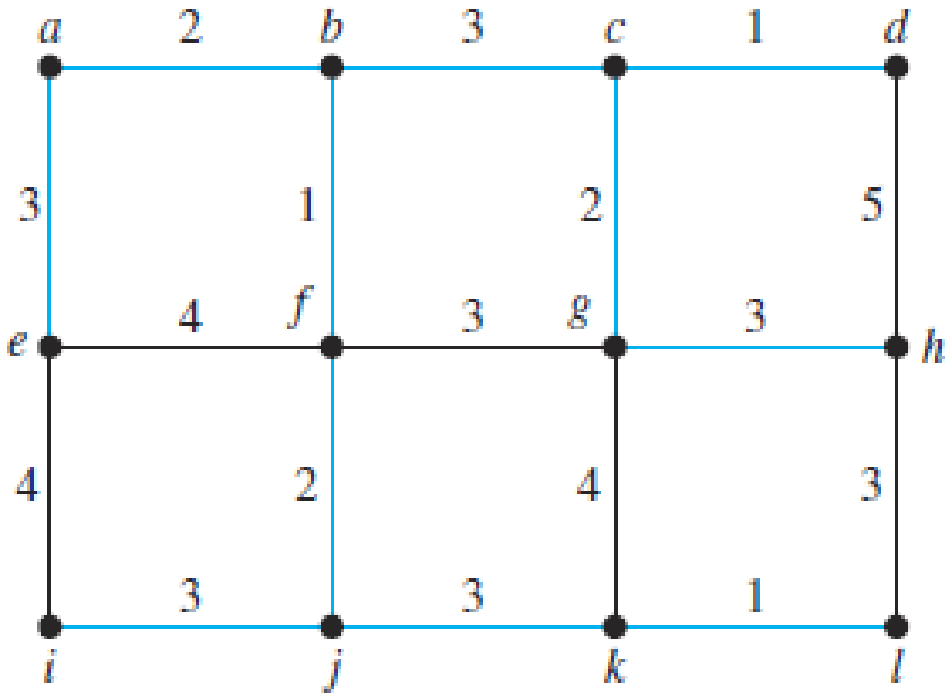
Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c, d\}$	$\{b, f\}$				



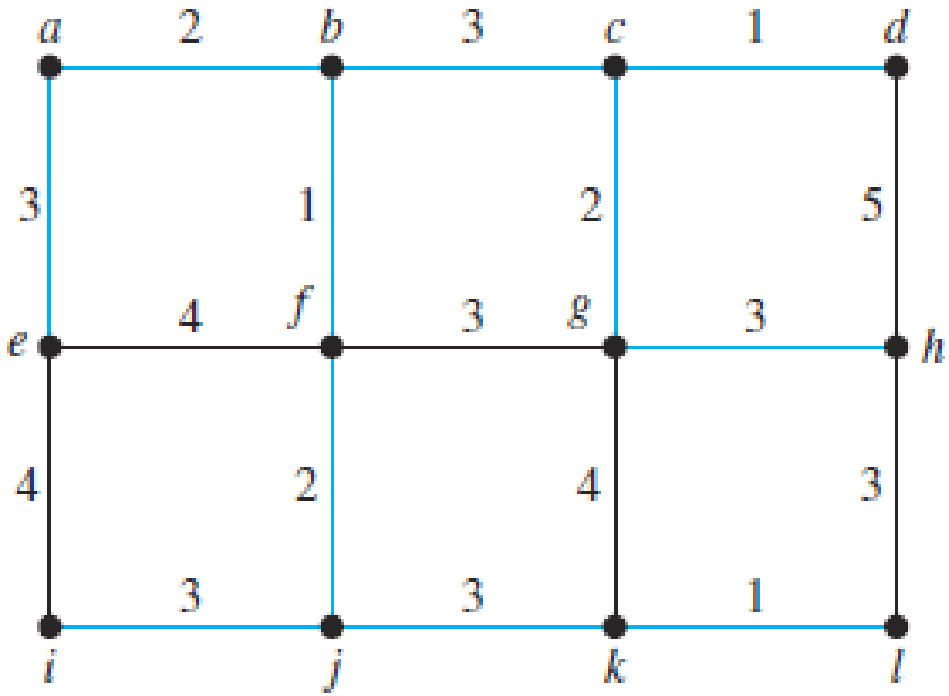
Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c, d\}$	$\{b, f\}$	$\{k, l\}$			



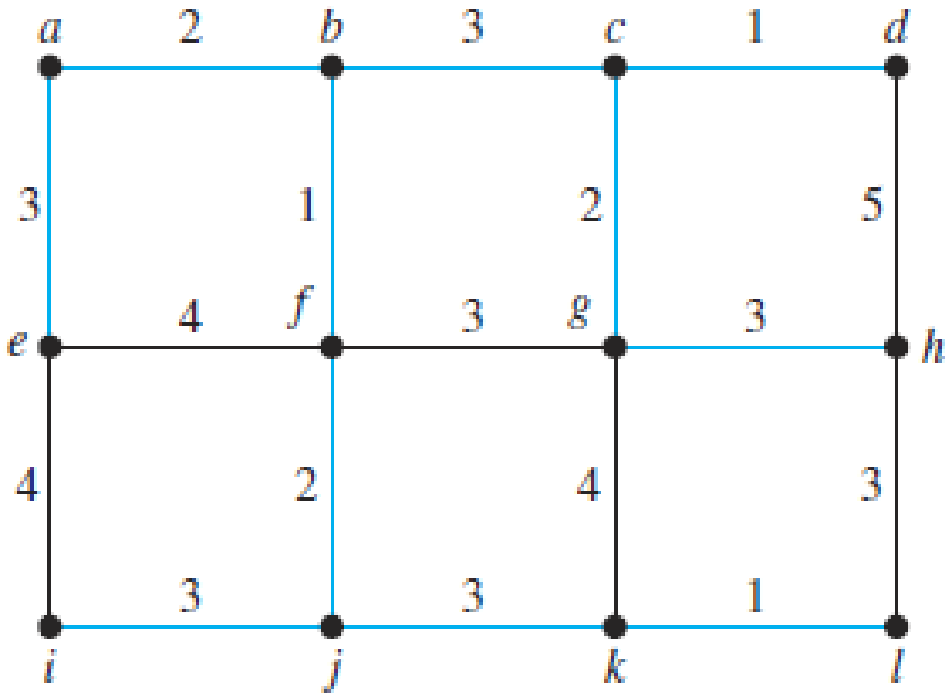
Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$		



Algoritma Kruskal bekerja sebagai berikut:

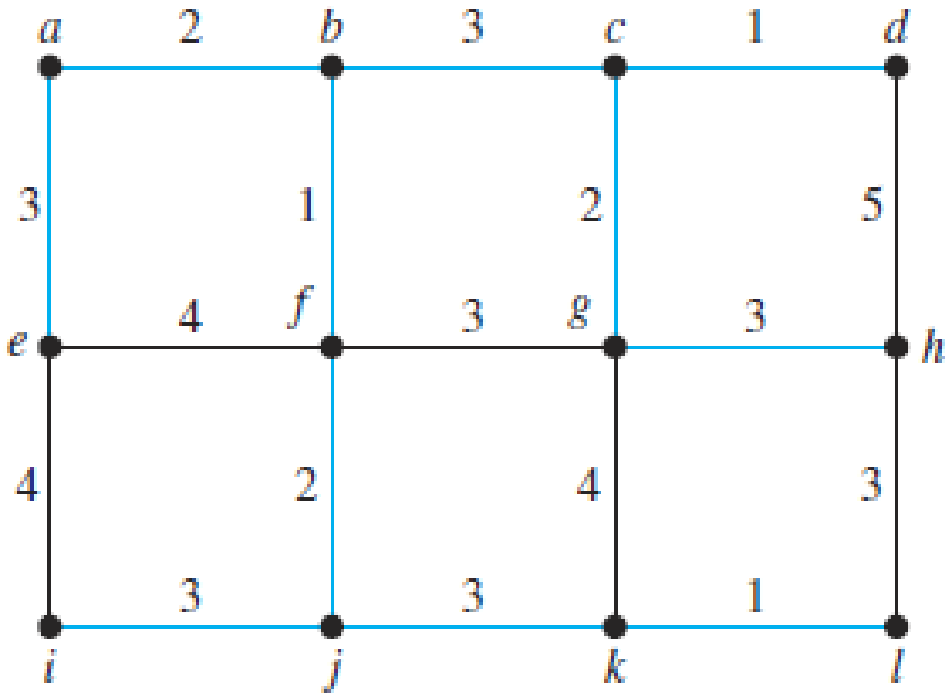
Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	



Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Bobot	1	1	1	2	2	2

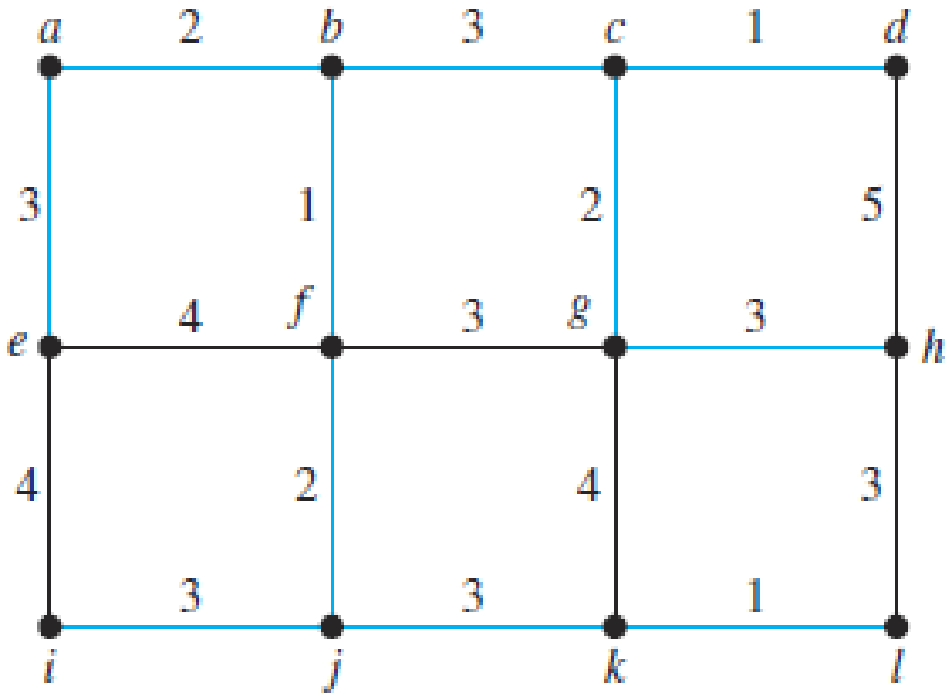
Pilihan ke-	7	8	9	10	11	Total
Sisi						



Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Bobot	1	1	1	2	2	2

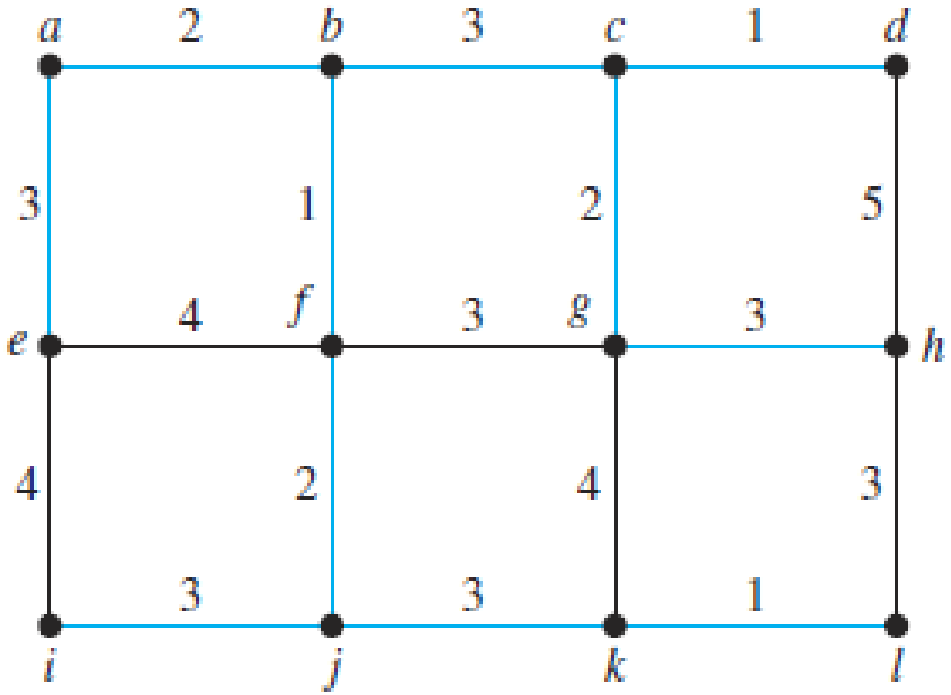
Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{a,e\}$					



Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Bobot	1	1	1	2	2	2

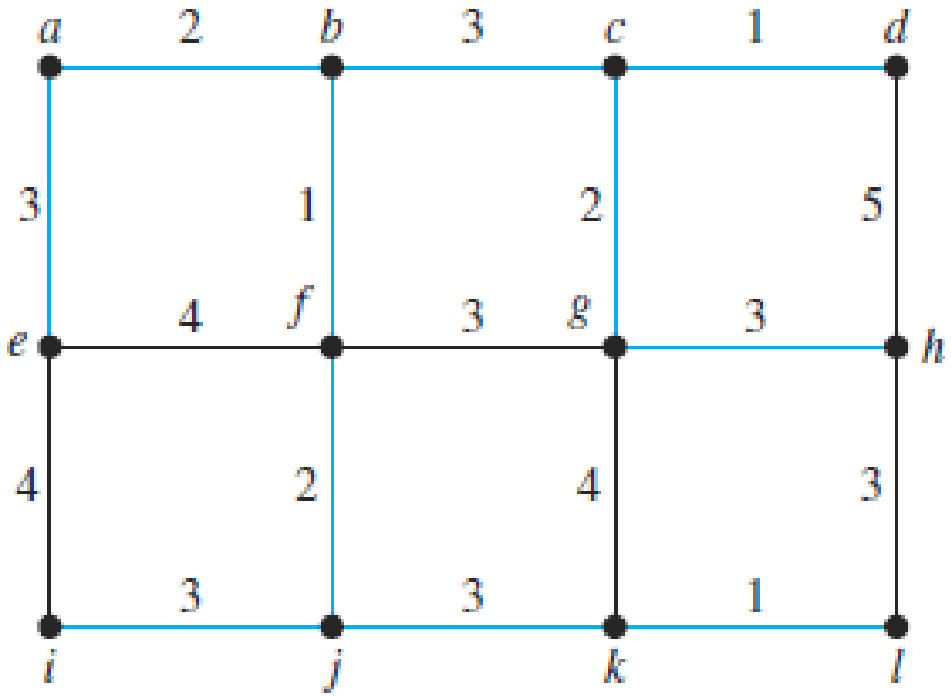
Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{a,e\}$	$\{b,c\}$				



Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Bobot	1	1	1	2	2	2

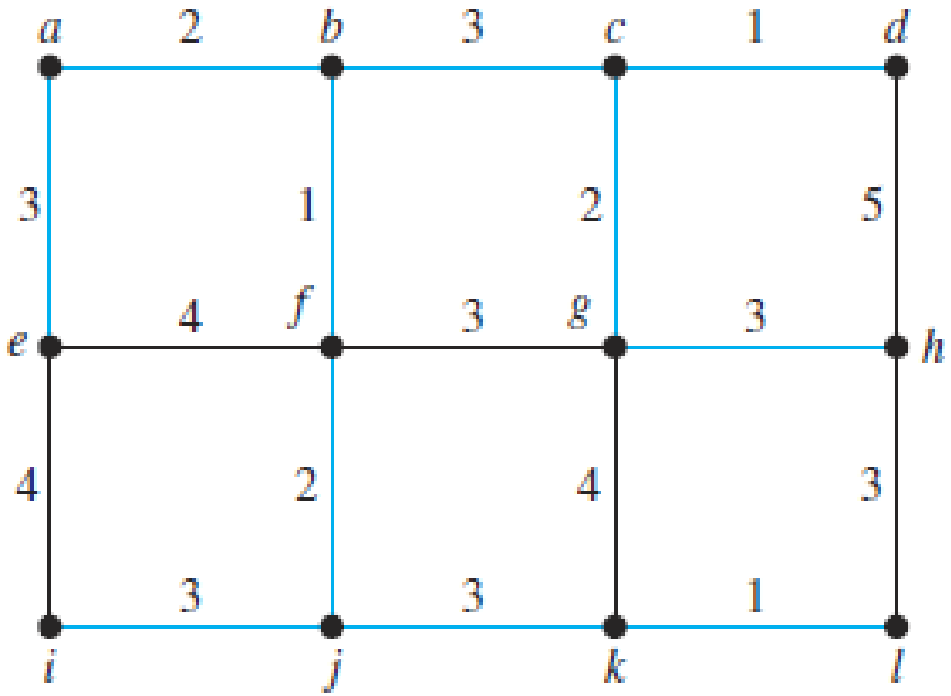
Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{a,e\}$	$\{b,c\}$	$\{g,h\}$			



Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Bobot	1	1	1	2	2	2

Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{a,e\}$	$\{b,c\}$	$\{g,h\}$	$\{i,j\}$		



Algoritma Kruskal bekerja sebagai berikut:

Pilihan ke-	1	2	3	4	5	6
Sisi	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Bobot	1	1	1	2	2	2

Pilihan ke-	7	8	9	10	11	Total
Sisi	$\{a,e\}$	$\{b,c\}$	$\{g,h\}$	$\{i,j\}$	$\{j,k\}$	
Bobot	3	3	3	3	3	24

Bahasan

Traversal pada Pohon (*Tree Traversal*)

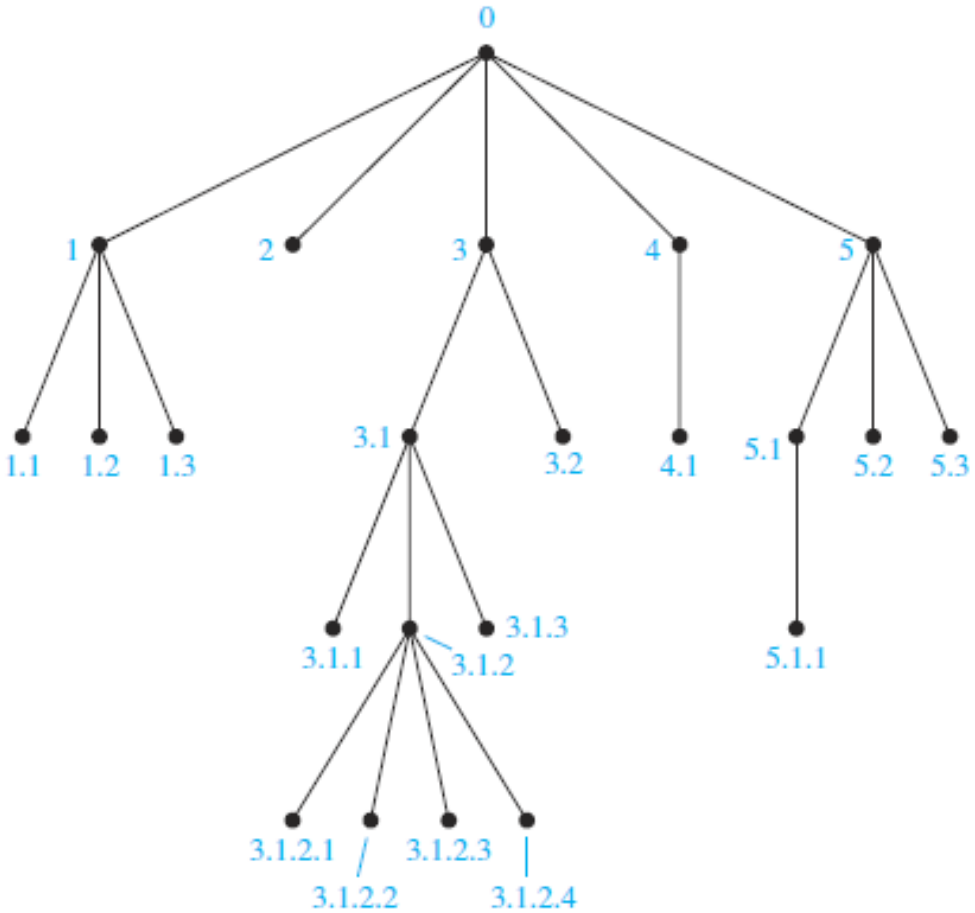
Pohon berakar (*rooted tree*) biasa dipakai untuk menyimpan informasi. Oleh karena itu kita memerlukan suatu metode untuk mengunjungi setiap simpul yang terdapat pada pohon tersebut. Proses pengunjungan ke setiap simpul tersebut dinamakan traversal (penjelajahan) pada pohon (*tree traversal*).

Sistem Alamat Universal (*Universal Address System*)

Sebuah pohon dapat digunakan untuk menyimpan informasi dengan sistem alamat universal (*universal address system*) yang merupakan pelabelan untuk setiap simpul pada sebuah pohon berakar. Pelabelan dilakukan secara rekursif sebagai berikut:

- Akar dengan dilabeli dengan 0, kemudian jika pada tingkat 1 terdapat k anak, maka setiap anak di tingkat 1 dilabeli dari kiri ke kanan dengan $1, 2, \ldots, k$.
- Untuk setiap simpul v di tingkat t dengan label A, jika v memiliki n anak, maka anak-anak v dilabeli dari kiri ke kanan dengan $A.1, A.2, \ldots, A.n$.

Contoh pemberian alamat universal.



Traversal Preorder (*Preorder Traversal*)

Traversal preorder dapat dijelaskan secara rekursif sebagai berikut.

Traversal Preorder

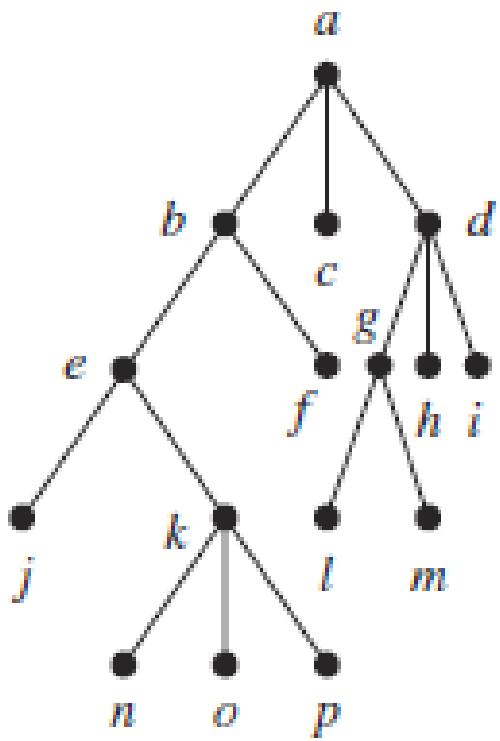
1. `procedure` $preorder\,(T\,(ordered\ root\ tree))$
2. $r :=$ akar dari T
3. `list` r
4. `for` setiap anak c dari r dengan urutan dari kiri ke kanan
5. $\quad T\,(c) :=$ subpohon dengan akar c
6. $\quad preorder\,(T\,(c))$

Secara intuitif, traversal preorder berkerja dengan prosedur:

1. mengunjungi akar,
2. mengunjungi subpohon bagian kiri, dan
3. mengunjungi subpohon bagian kanan.

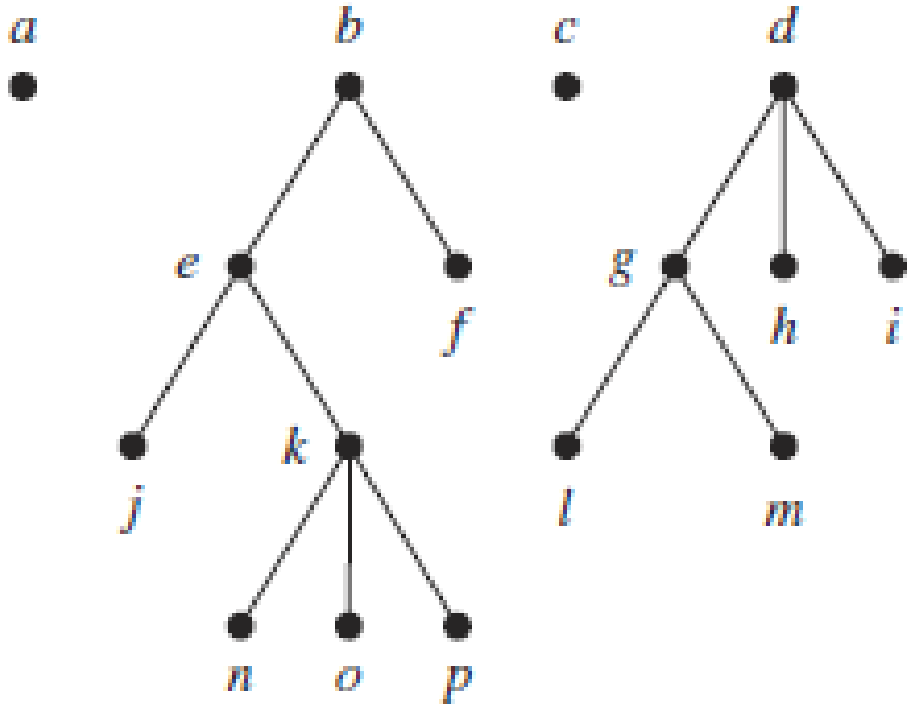
Contoh Traversal Preorder

Misalkan pohon yang simpulnya akan diurutkan adalah:

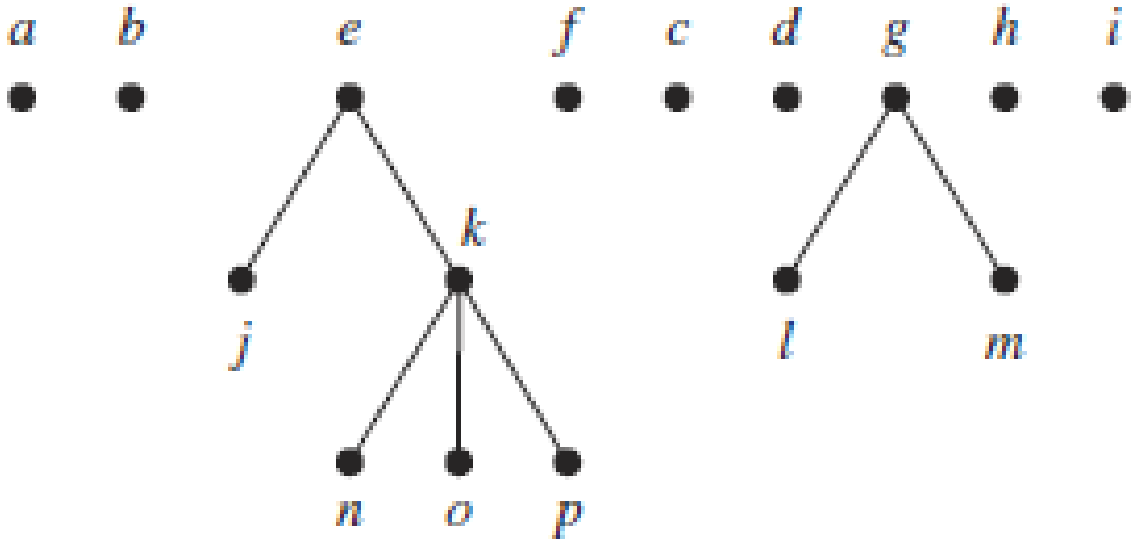


Preorder traversal: Visit root, visit subtrees left to right

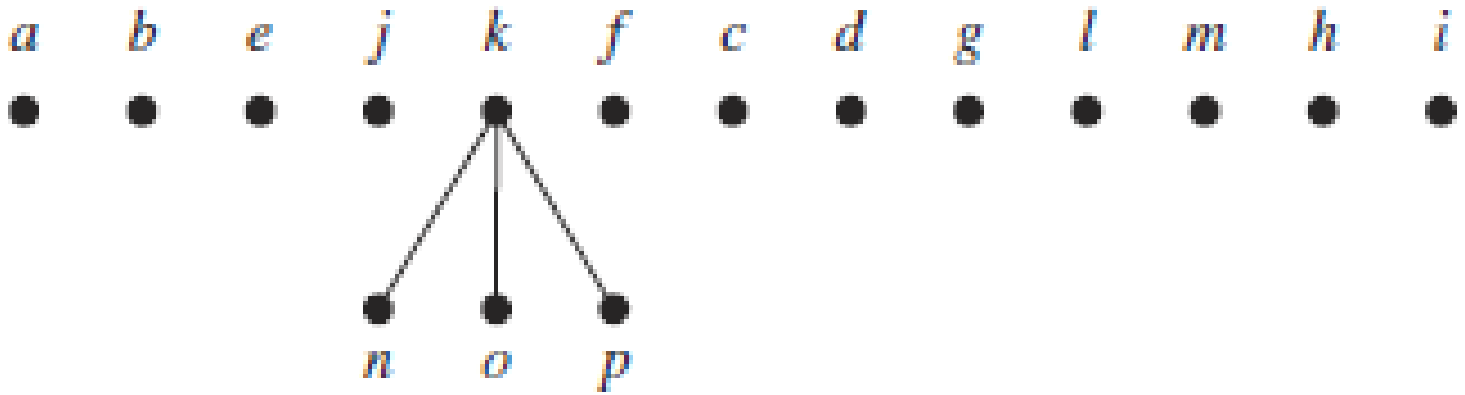
Iterasi pertama



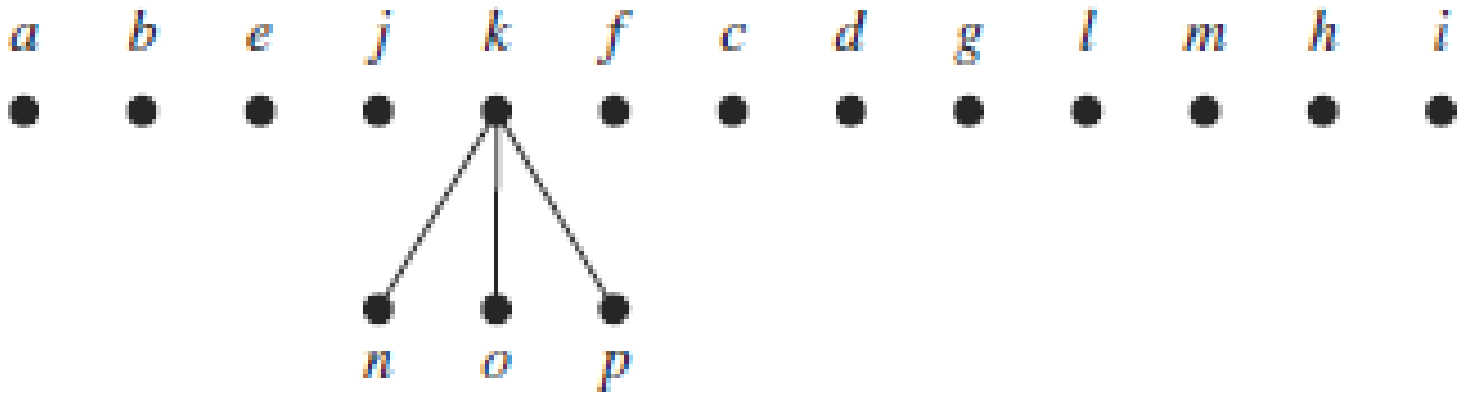
Iterasi kedua



Iterasi ketiga



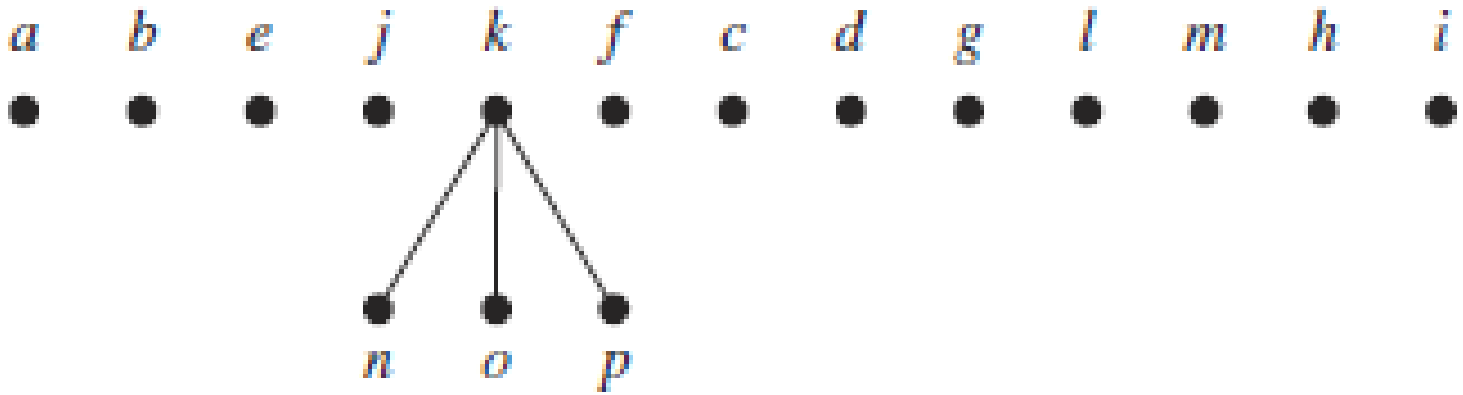
Iterasi ketiga



Iterasi keempat

Jadi urutan simpul dengan traversal preorder adalah

Iterasi ketiga



Iterasi keempat

Jadi urutan simpul dengan traversal preorder adalah
$a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i$.

Traversal Postorder (*Postorder Traversal*)

Traversal postorder dapat dijelaskan secara rekursif sebagai berikut.

Traversal Postorder

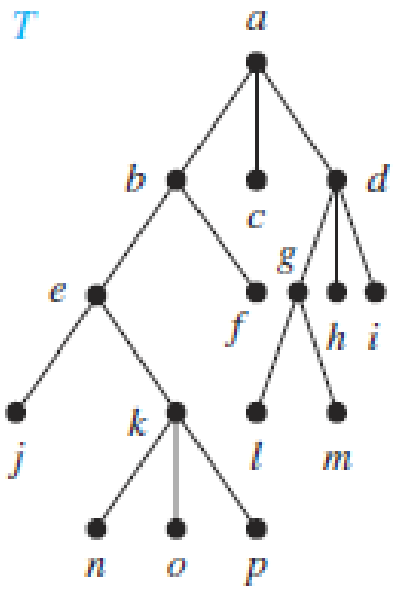
1. `procedure` $postorder\,(T\,(ordered\ root\ tree))$
2. $r :=$ akar dari T
3. `for` setiap anak c dari r dengan urutan dari kiri ke kanan
4. $\quad T\,(c) :=$ subpohon dengan akar c
5. $\quad postorder\,(T\,(c))$
6. `list` r

Secara intuitif, traversal postorder berkerja dengan prosedur:

1. mengunjungi subpohon dari kiri ke kanan, dan
2. mengunjungi akar.

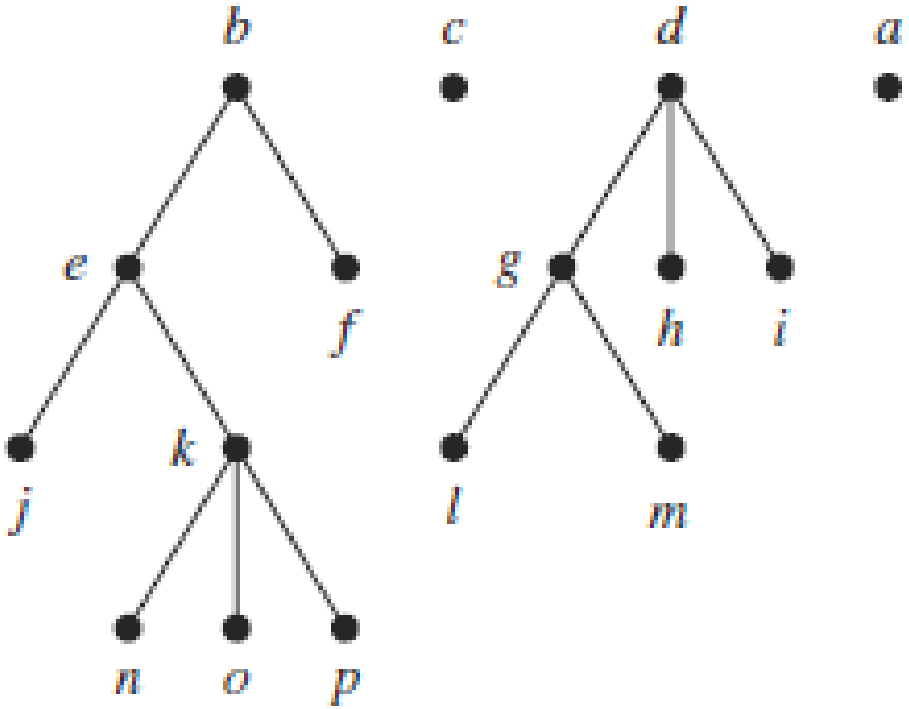
Contoh Traversal Postorder

Misalkan pohon yang simpulnya akan diurutkan adalah:

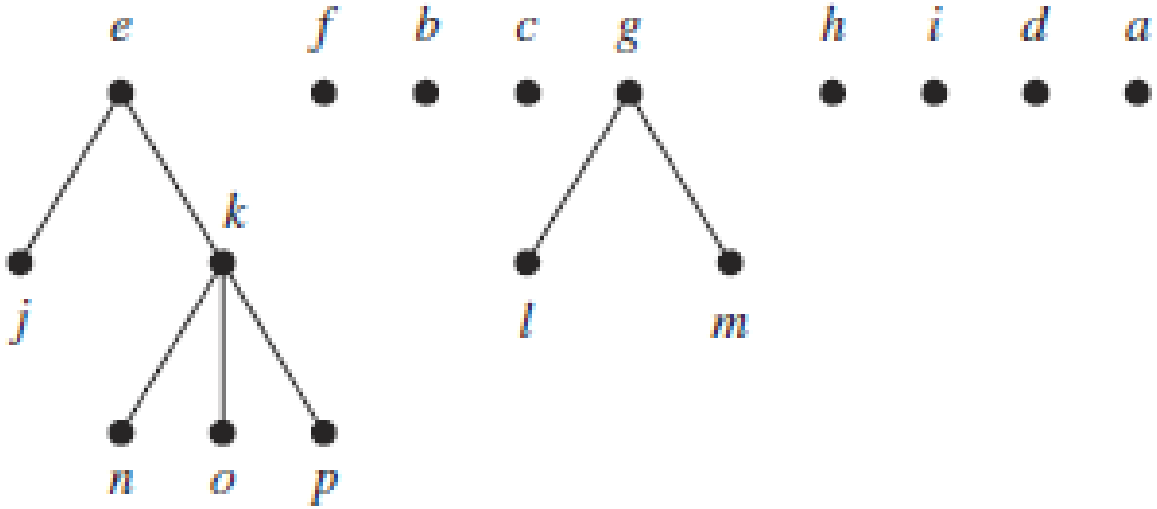


Postorder traversal: Visit subtrees left to right; visit root

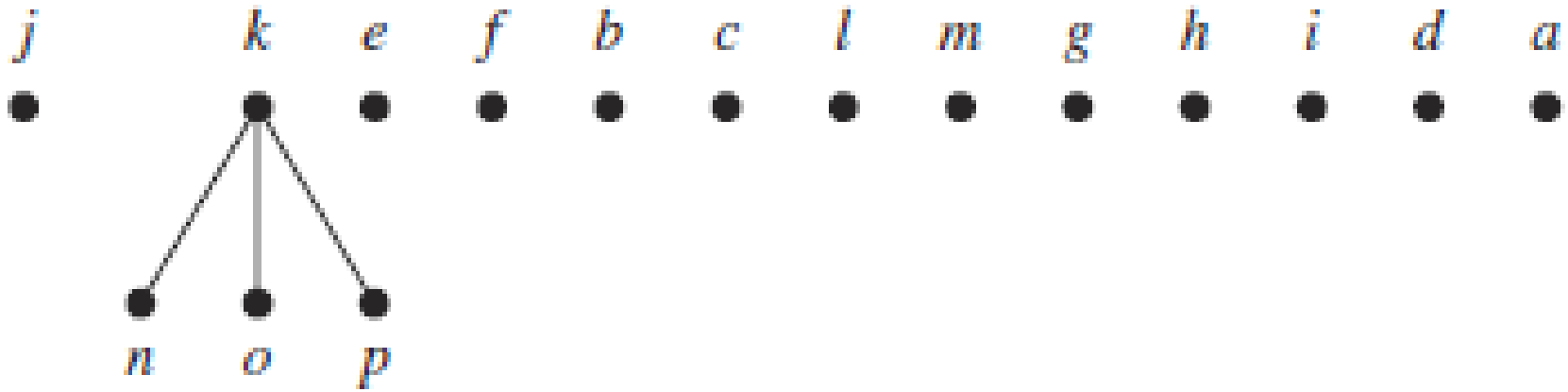
Iterasi pertama



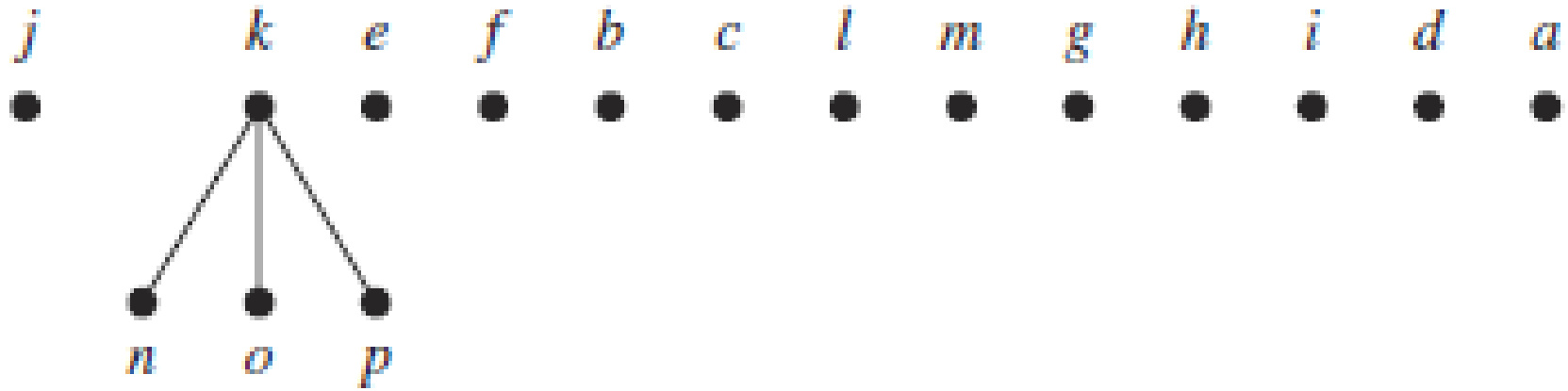
Iterasi kedua



Iterasi ketiga



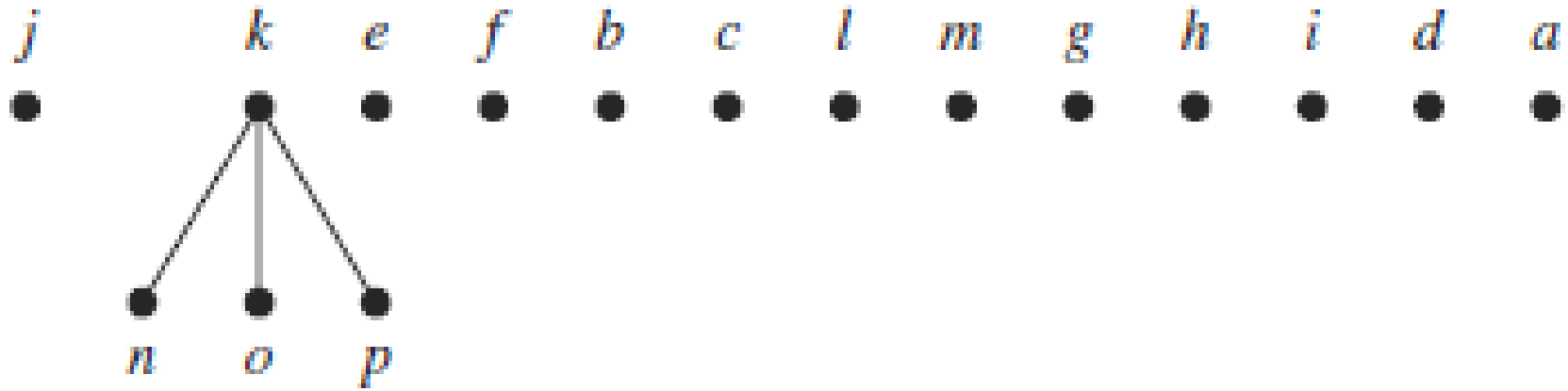
Iterasi ketiga



Iterasi keempat

Jadi urutan simpul dengan traversal postorder adalah

Iterasi ketiga



Iterasi keempat

Jadi urutan simpul dengan traversal postorder adalah
$j, n, o, p, k, e, f, b, c, l, m, g, h, i, d, a$.

Traversal Inorder (*Inorder Traversal*)

Traversal inorder dapat dijelaskan secara rekursif sebagai berikut.

Traversal Inorder

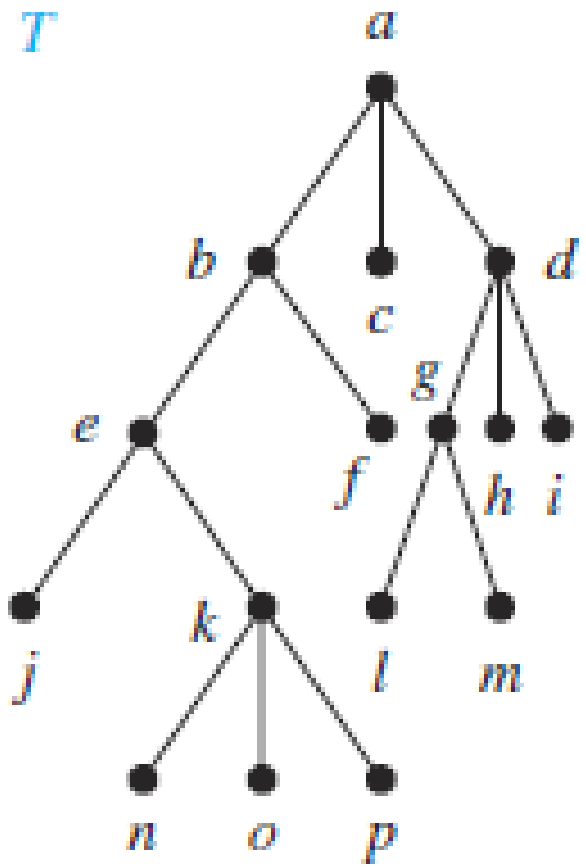
1. `procedure` $inorder\,(T\,(ordered\ root\ tree))$
2. $r :=$ akar dari T
3. `if` r adalah daun `then list` r
4. `else`
5. $\quad \ell :=$ anak pertama dari r dari kiri ke kanan
6. $\quad T\,(\ell) :=$ subpohon dengan akar ℓ
7. $\quad inorder\,(T\,(\ell))$
8. $\quad$ `list` r
9. $\quad$ `for` setiap anak dari c dari r kecuali ℓ dari kiri ke kanan
10. $\quad\quad T\,(c) :=$ subpohon dengan akar c
11. $\quad\quad inorder\,(T\,(c))$

Secara intuitif, traversal inorder berkerja dengan prosedur:

1. mengunjungi subpohon bagian paling kiri,
2. mengunjungi akar, dan
3. mengunjungi subpohon dari kiri ke kanan.

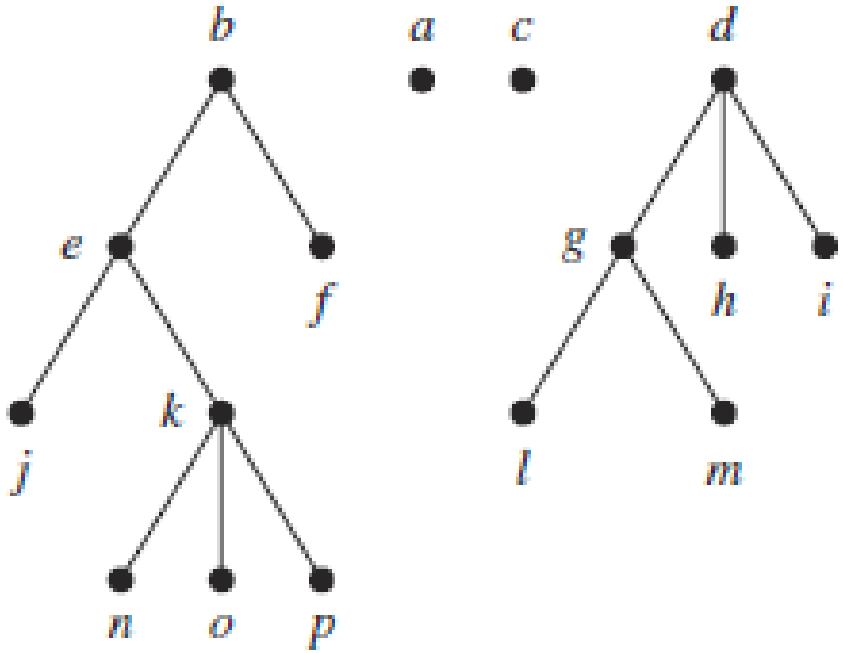
Contoh Traversal Inorder

Misalkan pohon yang simpulnya akan diurutkan adalah:

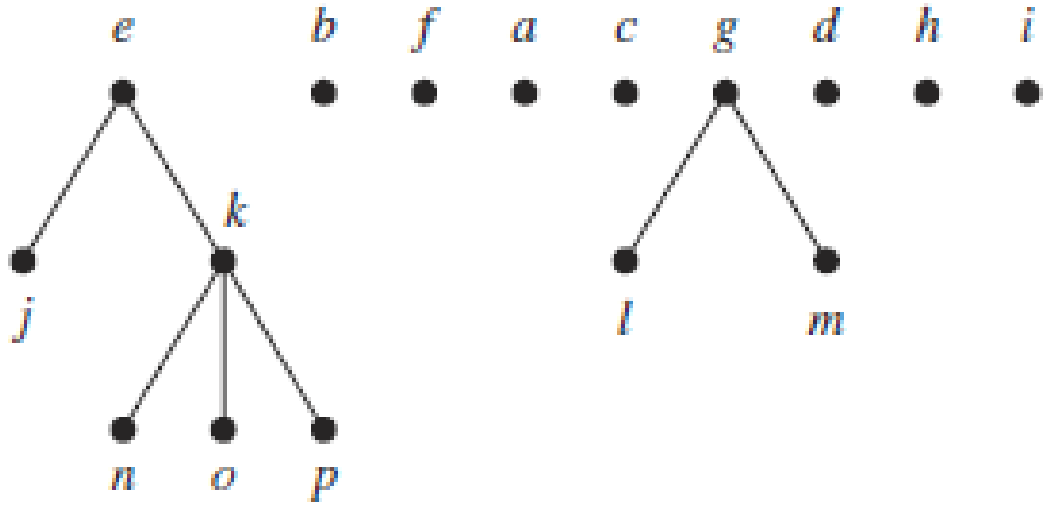


Inorder traversal: Visit leftmost subtree, visit root, visit other subtrees left to right

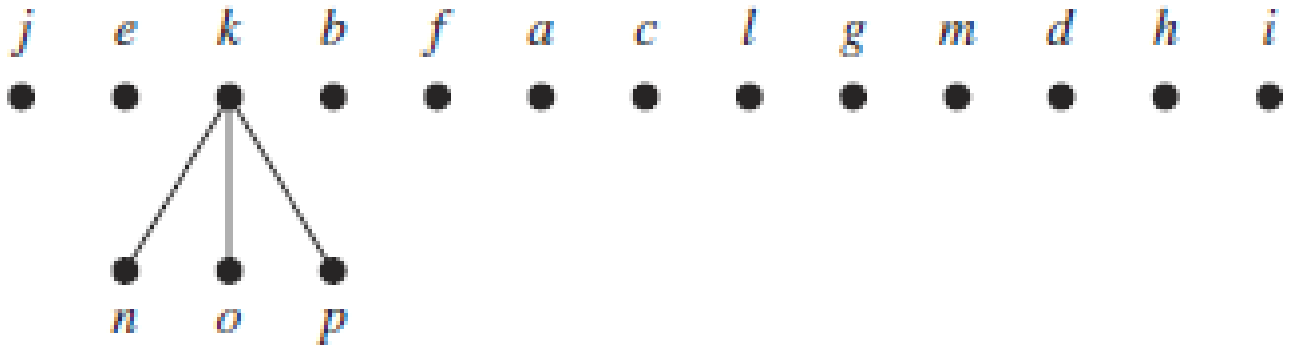
Iterasi pertama



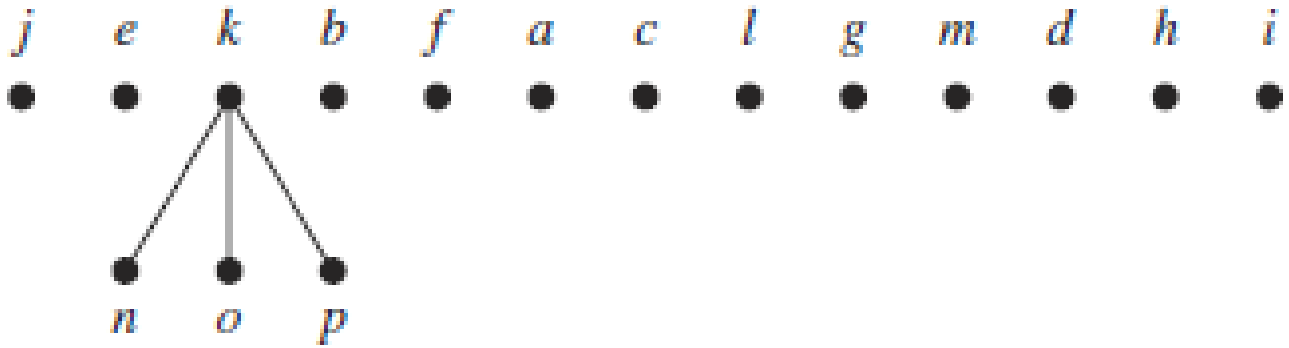
Iterasi kedua



Iterasi ketiga



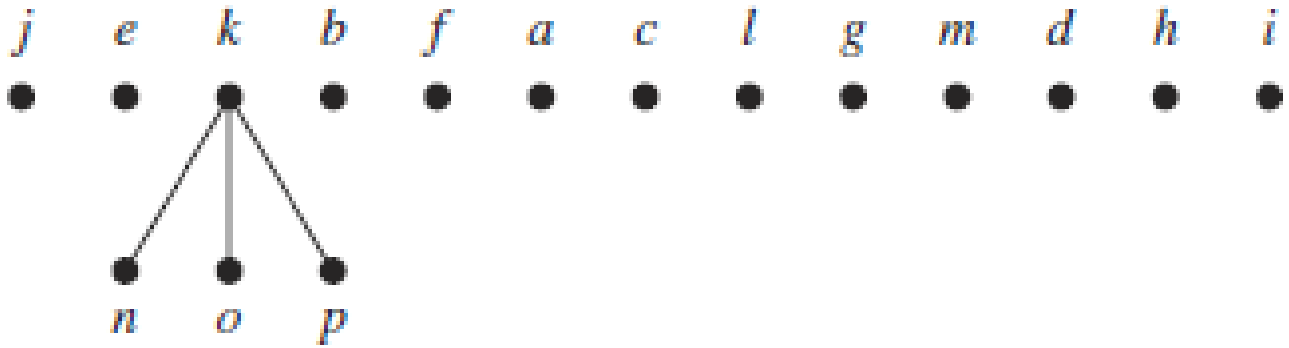
Iterasi ketiga



Iterasi keempat

Jadi urutan simpul dengan traversal inorder adalah

Iterasi ketiga



Iterasi keempat

Jadi urutan simpul dengan traversal inorder adalah
$j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i$.

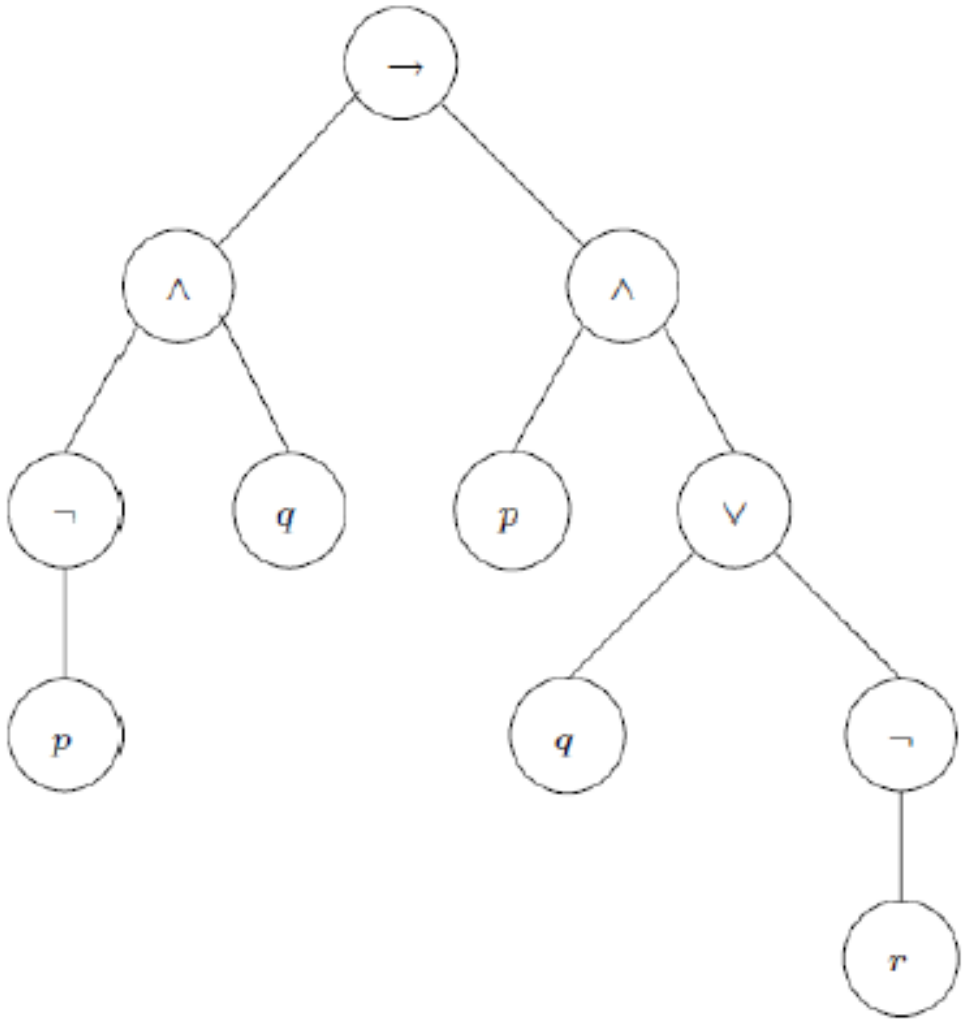
Bahasan

Pohon Urai (*Parse Tree*)

Pada materi kuliah Logika Matematika-A, kita telah mempelajari pohon urai untuk formula logika. Sebagai contoh, pohon urai untuk formula logika proposisi $(\neg p \wedge q) \rightarrow (p \wedge (q \vee \neg r))$ adalah

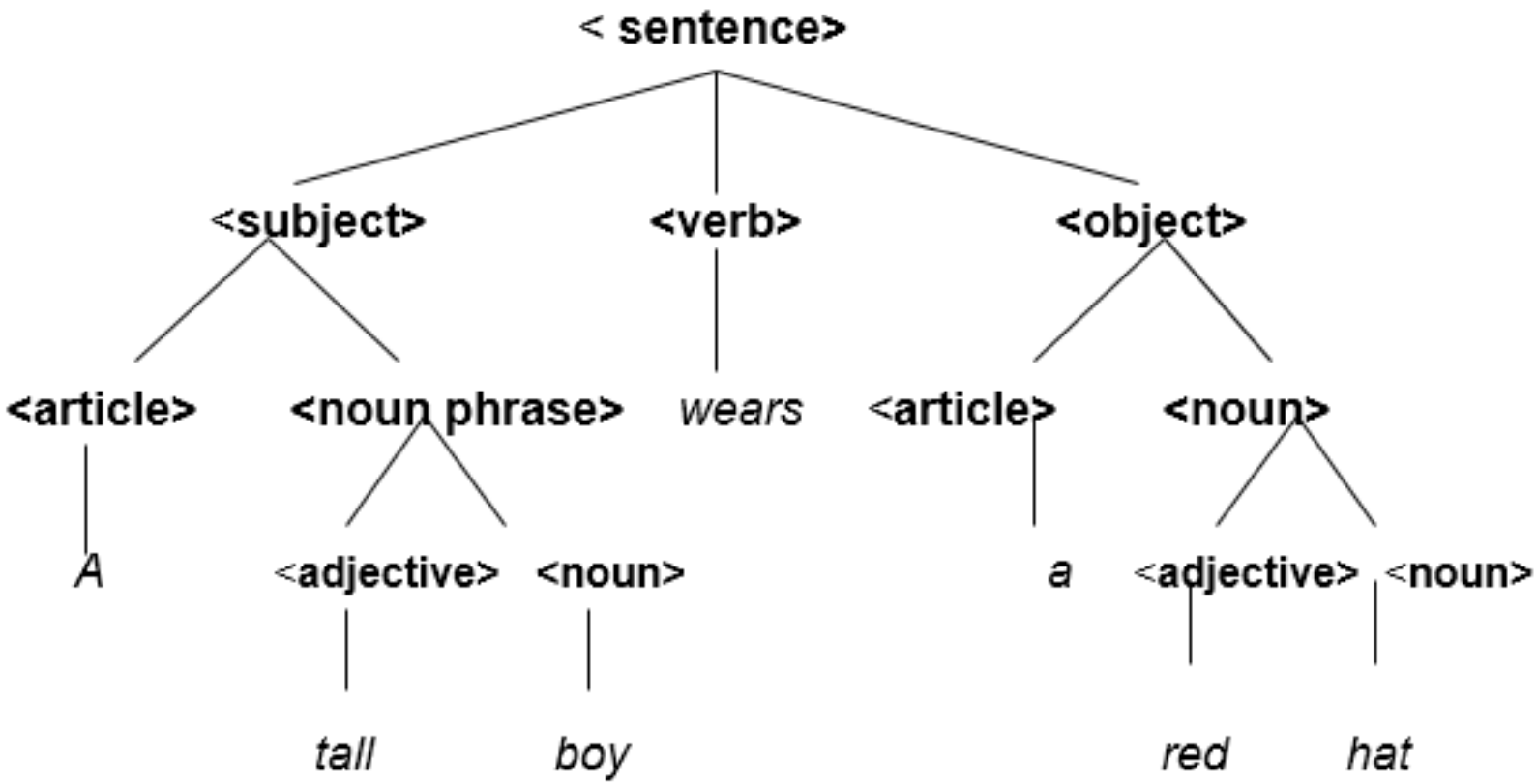
Pohon Urai (*Parse Tree*)

Pada materi kuliah Logika Matematika-A, kita telah mempelajari pohon urai untuk formula logika. Sebagai contoh, pohon urai untuk formula logika proposisi $(\neg p \wedge q) \rightarrow (p \wedge (q \vee \neg r))$ adalah



Pohon urai (*parse tree*) juga dapat dipakai untuk mengurai ekspresi matematika maupun kalimat dalam bahasa tertentu. Hal ini merupakan salah satu dasar dalam pemrosesan bahasa manusia (*natural language processing*).
Sebagai contoh, kalimat *"a tall boy wears a red hat"* dapat diuraikan sebagai berikut:

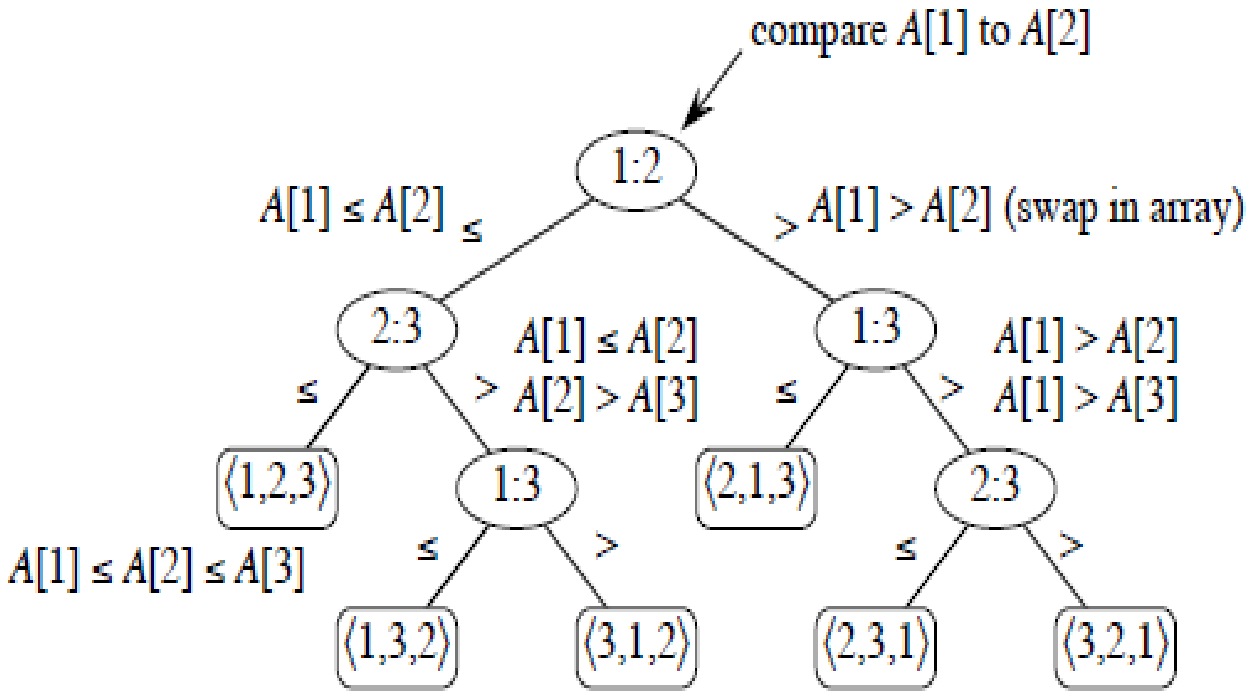
Pohon urai (*parse tree*) juga dapat dipakai untuk mengurai ekspresi matematika maupun kalimat dalam bahasa tertentu. Hal ini merupakan salah satu dasar dalam pemrosesan bahasa manusia (*natural language processing*).
Sebagai contoh, kalimat *"a tall boy wears a red hat"* dapat diuraikan sebagai berikut:



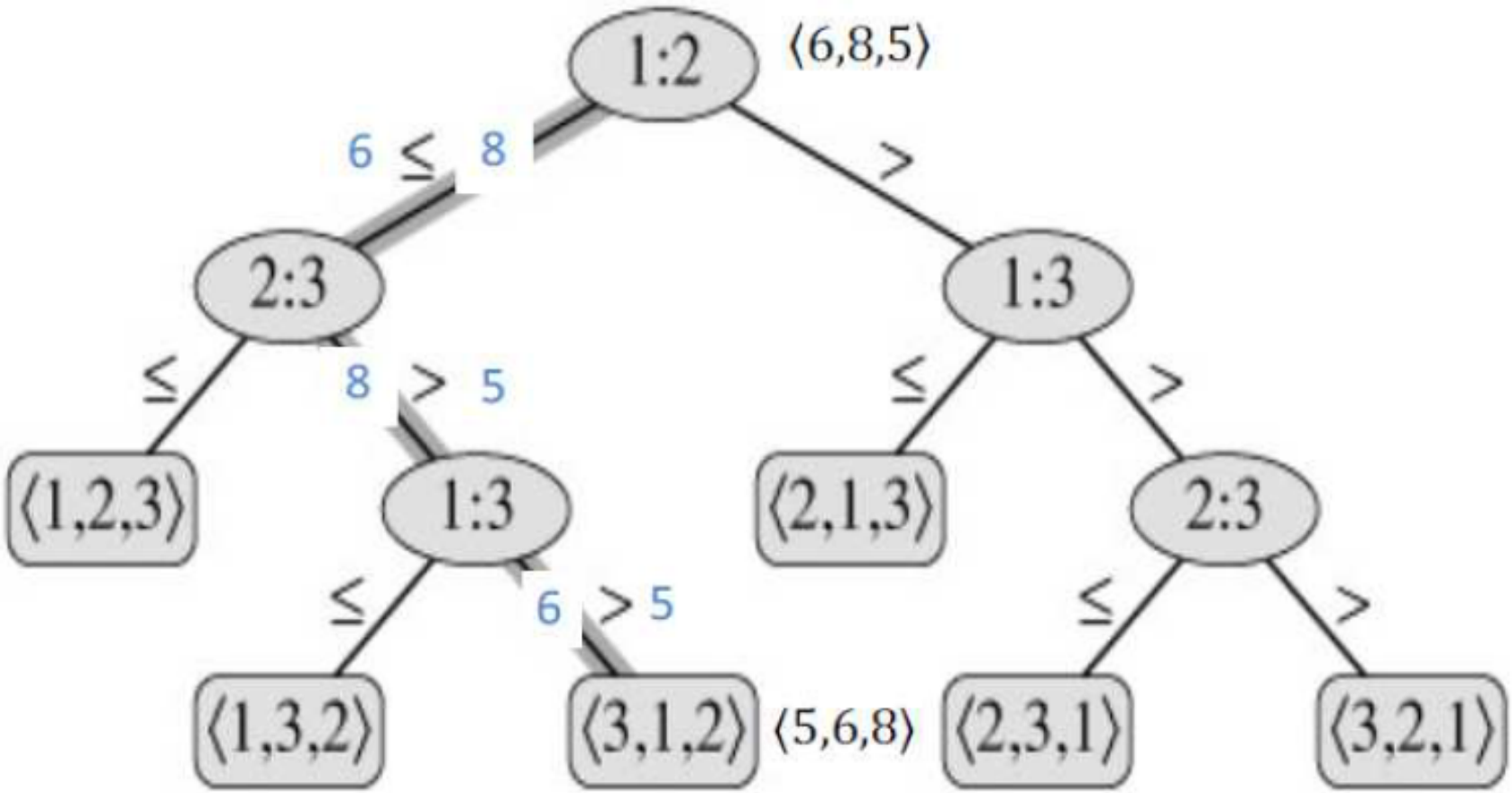
Bahasan

Pohon Keputusan (*Decision Tree*)

Pohon keputusan (*decision tree*) merupakan pohon yang menggambarkan cara pengambilan keputusan pada suatu algoritma tertentu. Misalkan kita memiliki sebuah array $A = \langle A[1], A[2], A[3] \rangle$. Algoritma pengurutan elemen-elemen *array* tersebut dengan urutan menaik (*ascending*) dapat digambarkan dalam pohon berikut.



Sebagai contoh, berikut adalah ilustrasi proses pengurutan array tiga elemen $\langle 6, 8, 5 \rangle$



Bahasan

Ekspresi Matematika

Di sekolah menengah kita sudah mengenal ekspresi matematika yang kompleks dan melibatkan lebih dari satu operator, contohnya $2 + 4 \times 5$. Nilai manakah yang benar:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

Ketika disimpan dalam komputer, sebuah ekspresi matematika setidaknya dapat dinyatakan dalam tiga cara, yaitu:

Ekspresi Matematika

Di sekolah menengah kita sudah mengenal ekspresi matematika yang kompleks dan melibatkan lebih dari satu operator, contohnya $2 + 4 \times 5$. Nilai manakah yang benar:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

Ketika disimpan dalam komputer, sebuah ekspresi matematika setidaknya dapat dinyatakan dalam tiga cara, yaitu:

- notasi infiks/ *infix notation* (notasi standar yang paling sering digunakan dalam kehidupan manusia sehari-hari),

Ekspresi Matematika

Di sekolah menengah kita sudah mengenal ekspresi matematika yang kompleks dan melibatkan lebih dari satu operator, contohnya $2 + 4 \times 5$. Nilai manakah yang benar:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

Ketika disimpan dalam komputer, sebuah ekspresi matematika setidaknya dapat dinyatakan dalam tiga cara, yaitu:

- notasi infiks/ *infix notation* (notasi standar yang paling sering digunakan dalam kehidupan manusia sehari-hari),
- notasi prefiks/ *prefix notation* (menggunakan traversal preorder), dan

Ekspresi Matematika

Di sekolah menengah kita sudah mengenal ekspresi matematika yang kompleks dan melibatkan lebih dari satu operator, contohnya $2 + 4 \times 5$. Nilai manakah yang benar:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

Ketika disimpan dalam komputer, sebuah ekspresi matematika setidaknya dapat dinyatakan dalam tiga cara, yaitu:

- notasi infiks/ *infix notation* (notasi standar yang paling sering digunakan dalam kehidupan manusia sehari-hari),
- notasi prefiks/ *prefix notation* (menggunakan traversal preorder), dan
- notasi postfiks/ *postfix notation* (menggunakan traversal postorder).

Notasi prefiks juga dikenal dengan istilah notasi Polandia (*Polish notation*) dan notasi postfiks juga dikenal dengan istilah notasi Polandia terbalik (*reverse Polish notation*).

Presedens (Hirarki) Operator Aritmetika Dasar

Presedens operator aritmetika memberikan suatu aturan operator mana yang harus lebih dulu dioperasikan (dikenakan pada suatu *operand*).

Presedens (Hirarki) Operator Aritmetika Dasar

Presedens operator aritmetika memberikan suatu aturan operator mana yang harus lebih dulu dioperasikan (dikenakan pada suatu *operand*).

Tabel urutan pengerjaan (presendens) operator aritmetika dasar.

Operator	Urutan
$\hat{}$ (pangkat)	1
$*$ (perkalian)	2
$/$ (pembagian)	3
$+$ (penjumlahan)	4
$-$ (pengurangan)	5

Presedens (Hirarki) Operator Aritmetika Dasar

Presedens operator aritmetika memberikan suatu aturan operator mana yang harus lebih dulu dioperasikan (dikenakan pada suatu *operand*).

Tabel urutan pengerjaan (presendens) operator aritmetika dasar.

Operator	Urutan
$\hat{}$ (pangkat)	1
$*$ (perkalian)	2
$/$ (pembagian)	3
$+$ (penjumlahan)	4
$-$ (pengurangan)	5

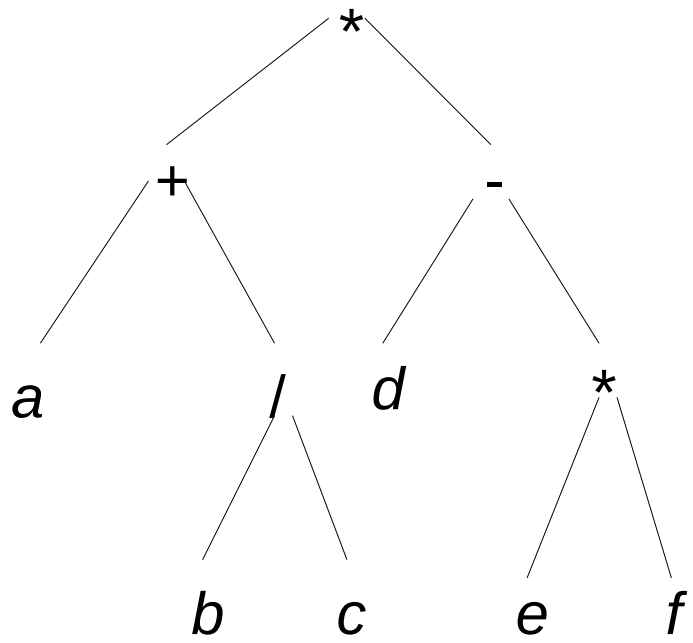
Sebagaimana telah dipelajari di sekolah menengah, kita dapat menggunakan tanda kurung "(" dan ")" untuk memperjelas operasi yang harus didahulukan.

Contoh

Misalkan kita memiliki ekspresi matematika $(a + b/c) * (d - e * f)$. Pohon urai (*parse tree*) untuk ekspresi ini adalah:

Contoh

Misalkan kita memiliki ekspresi matematika $(a + b/c) * (d - e * f)$. Pohon urai (*parse tree*) untuk ekspresi ini adalah:

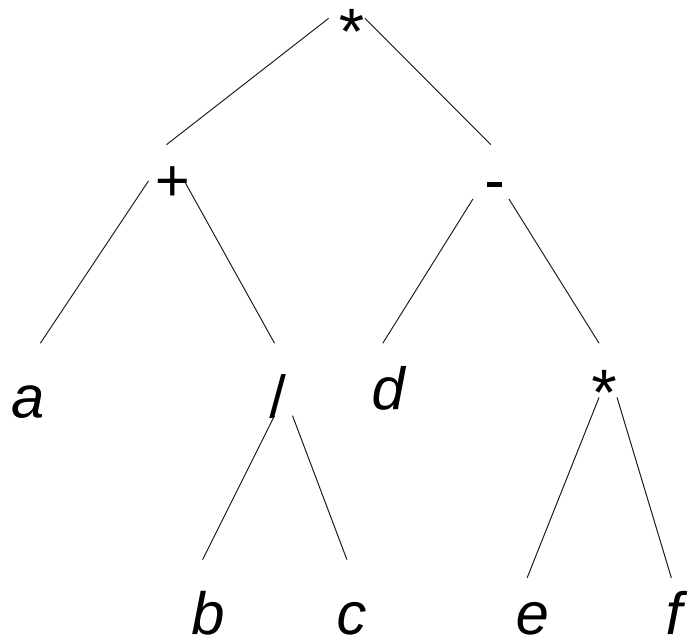


Ekspresi $(a + b/c) * (d - e * f)$ dinyatakan dalam notasi infiks. Kita dapat memperoleh notasi prefiks dan postfiks dari ekspresi ini dengan memanfaatkan traversal preorder dan postorder sebelumnya, sehingga diperoleh:

- notasi prefiksnya:

Contoh

Misalkan kita memiliki ekspresi matematika $(a + b/c) * (d - e * f)$. Pohon urai (*parse tree*) untuk ekspresi ini adalah:

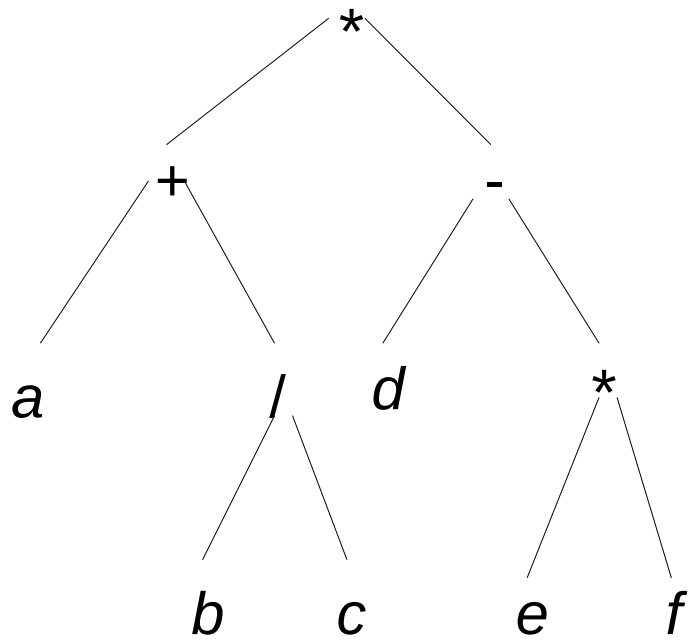


Ekspresi $(a + b/c) * (d - e * f)$ dinyatakan dalam notasi infiks. Kita dapat memperoleh notasi prefiks dan postfiks dari ekspresi ini dengan memanfaatkan traversal preorder dan postorder sebelumnya, sehingga diperoleh:

- notasi prefiksnya: $* + a/bc - d * ef$ (hasil traversal preorder dari pohon urai);
- notasi postfiksnya:

Contoh

Misalkan kita memiliki ekspresi matematika $(a + b/c) * (d - e * f)$. Pohon urai (*parse tree*) untuk ekspresi ini adalah:



Ekspresi $(a + b/c) * (d - e * f)$ dinyatakan dalam notasi infiks. Kita dapat memperoleh notasi prefiks dan postfiks dari ekspresi ini dengan memanfaatkan traversal preorder dan postorder sebelumnya, sehingga diperoleh:

- notasi prefiksnya: $* + a/bc - d * ef$ (hasil traversal preorder dari pohon urai);
- notasi postfiksnya: $abc/ + def * -*$ (hasil traversal postorder dari pohon urai).

Kelebihan Notasi Prefiks dan Postfiks

Meskipun terkesan agak sulit dibaca oleh manusia, notasi prefiks dan postfiks memiliki kelebihan, yaitu kedua notasi ini tidak memerlukan tanda kurung untuk menghindari ambiguitas. Oleh karenanya kedua notasi ini banyak dipakai dalam desain dan pembuatan *compiler*.