Algoritme dan Struktur Data

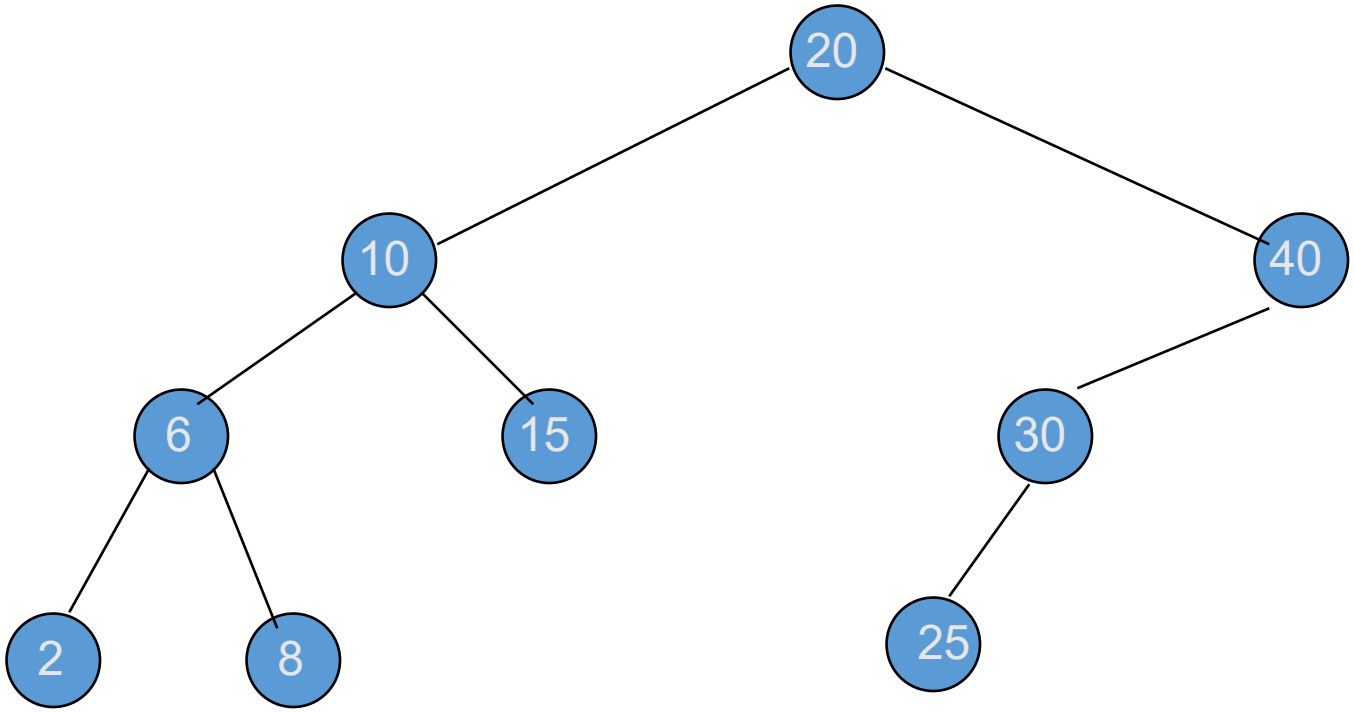
Binary Search Tree (BST)

Putra Pandu Adikara
Fakultas Ilmu Komputer
Universitas Brawijaya

Definisi

- Sebuah *binary tree* yang mana *subtree* sebelah kiri lebih kecil dari *subtree* sebelah kanan.

Contoh BST



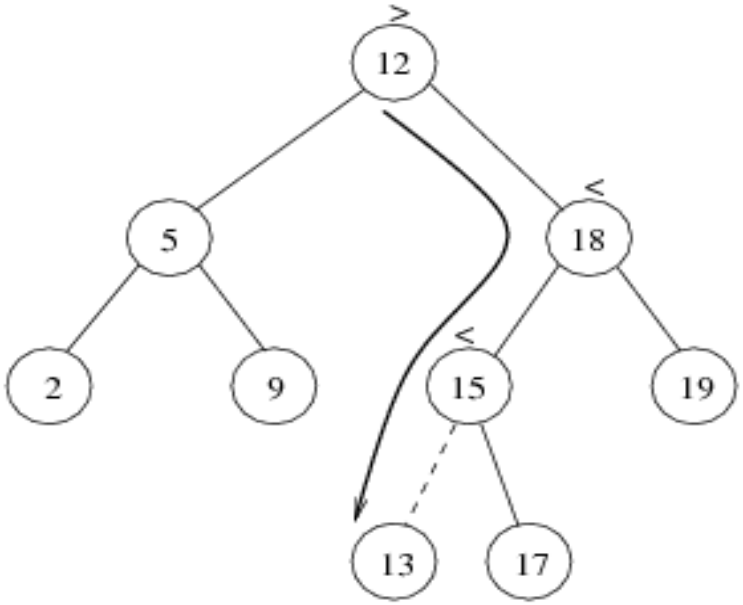
Binary Search Tree

- Operasi BST:
  - penambahan, penghapusan, pencarian node tertentu, pencarian niai terkecil dan pencarian nilai terbesar.

- Properti Binary Search Tree:
  - Untuk setiap node X, semua elemen di subpohon kirinya bernilai lebih kecil dari nilai X dan semua elemen di subpohon kanannya bernilai lebih besar dari nilai X.

Insert

- Dimulai dengan penelusuran dari root untuk mencari posisi yang tepat.
- Jika elemen X ditemukan (berarti X sudah ada di BST), maka tidak perlu melakukan aksi apapun.
- Jika tidak, maka letakkan X sebagai node terakhir pada jalurr penelusuran



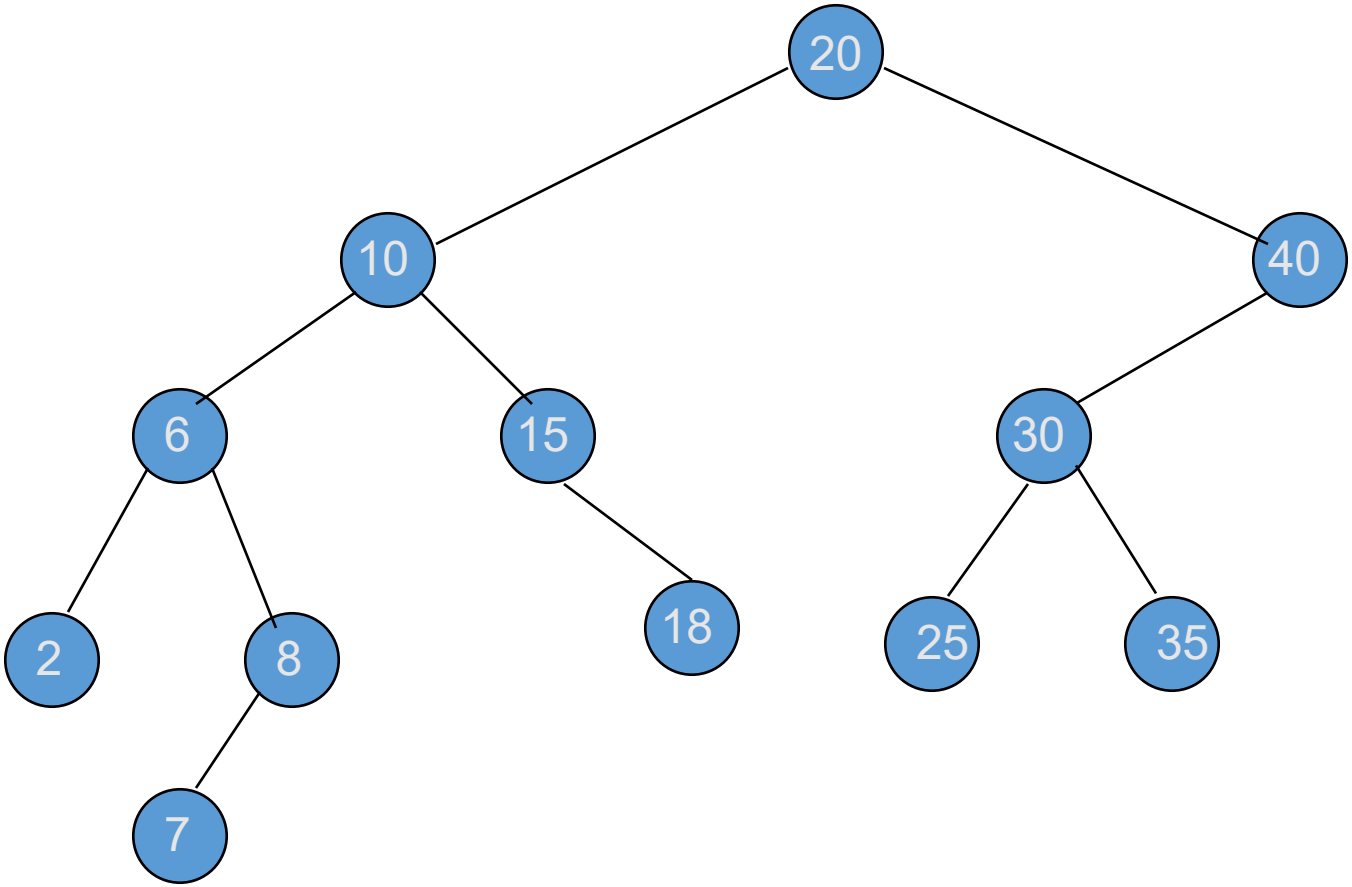
- Time complexity = O(height of the tree)

Delete

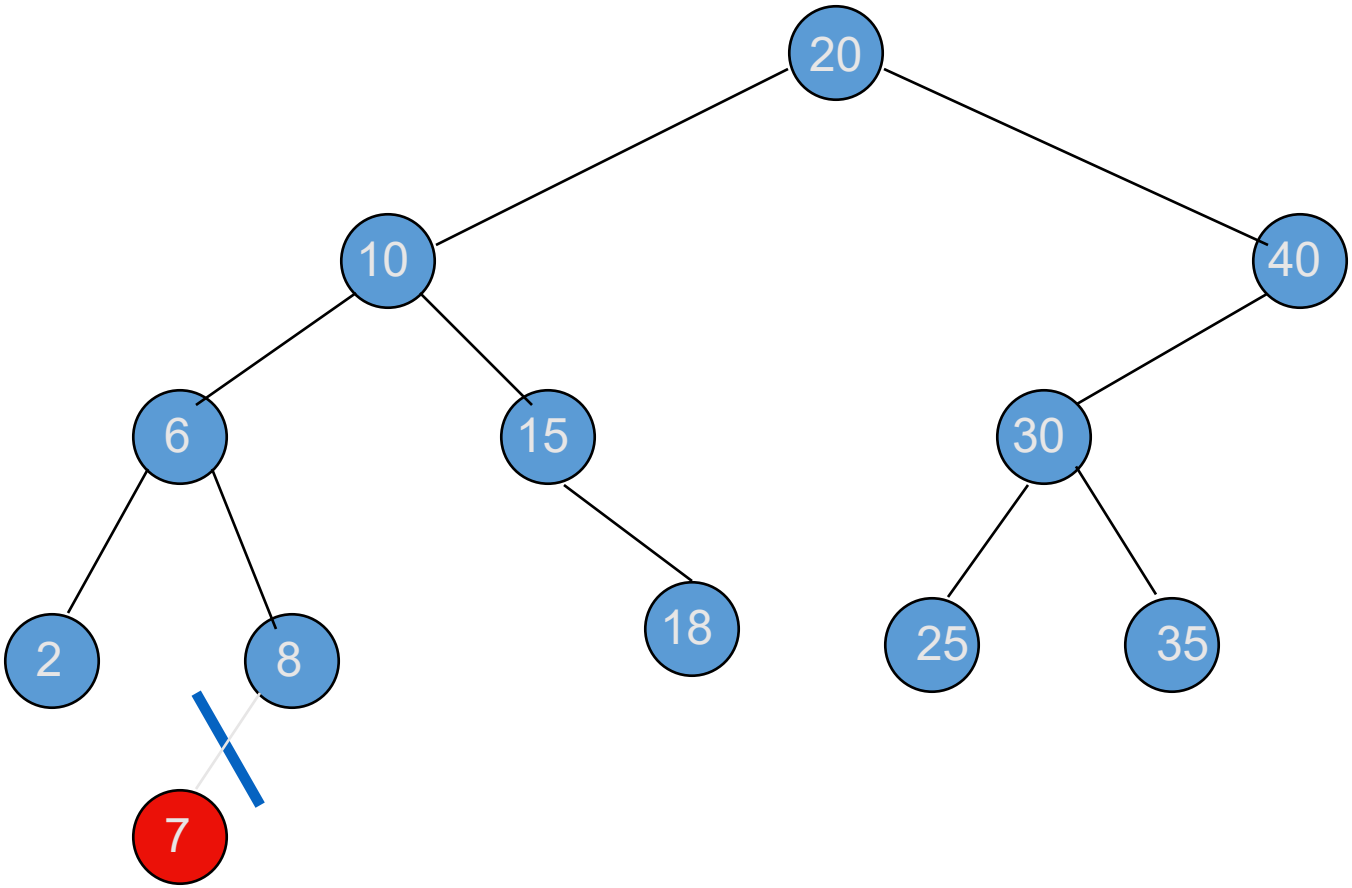
- Saat akan menghapus sebuah node, kita juga harus memikirkan seluruh node anak dari node tsb.
  - Hal penting adalah agar pohon setelah dihapus tetap merupakan BST.

Operasi Penghapusan / remove()

- Ada 3 kasus:
  - Elemen ada di leaf/daun.
  - Elemen yang memiliki degree 1.
  - Elemen yang memiliki degree 2.

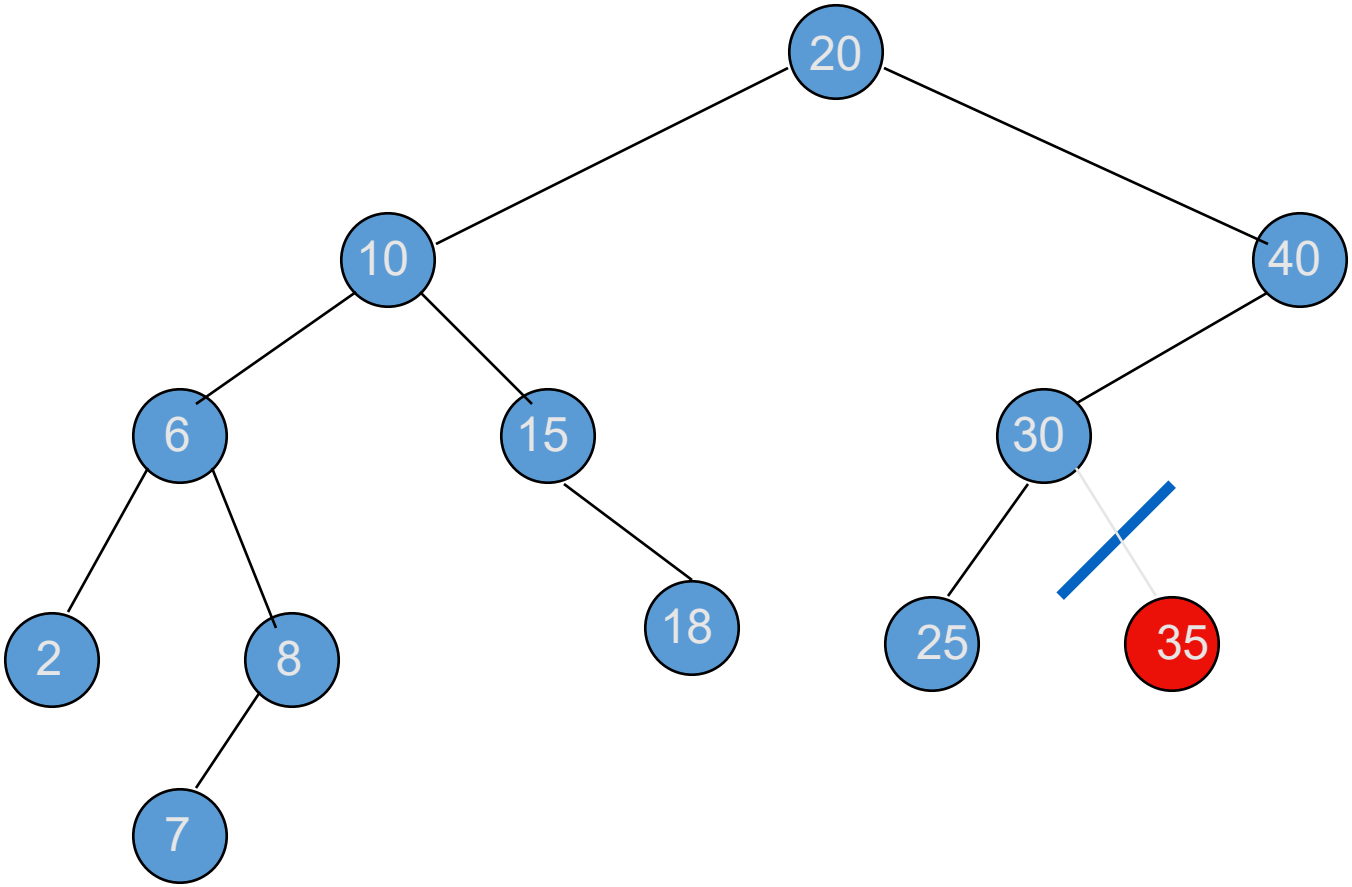

20
10
40
6
15
30
2
8
18
25
35
7

1. Penghapusan Node Daun (Node 7)



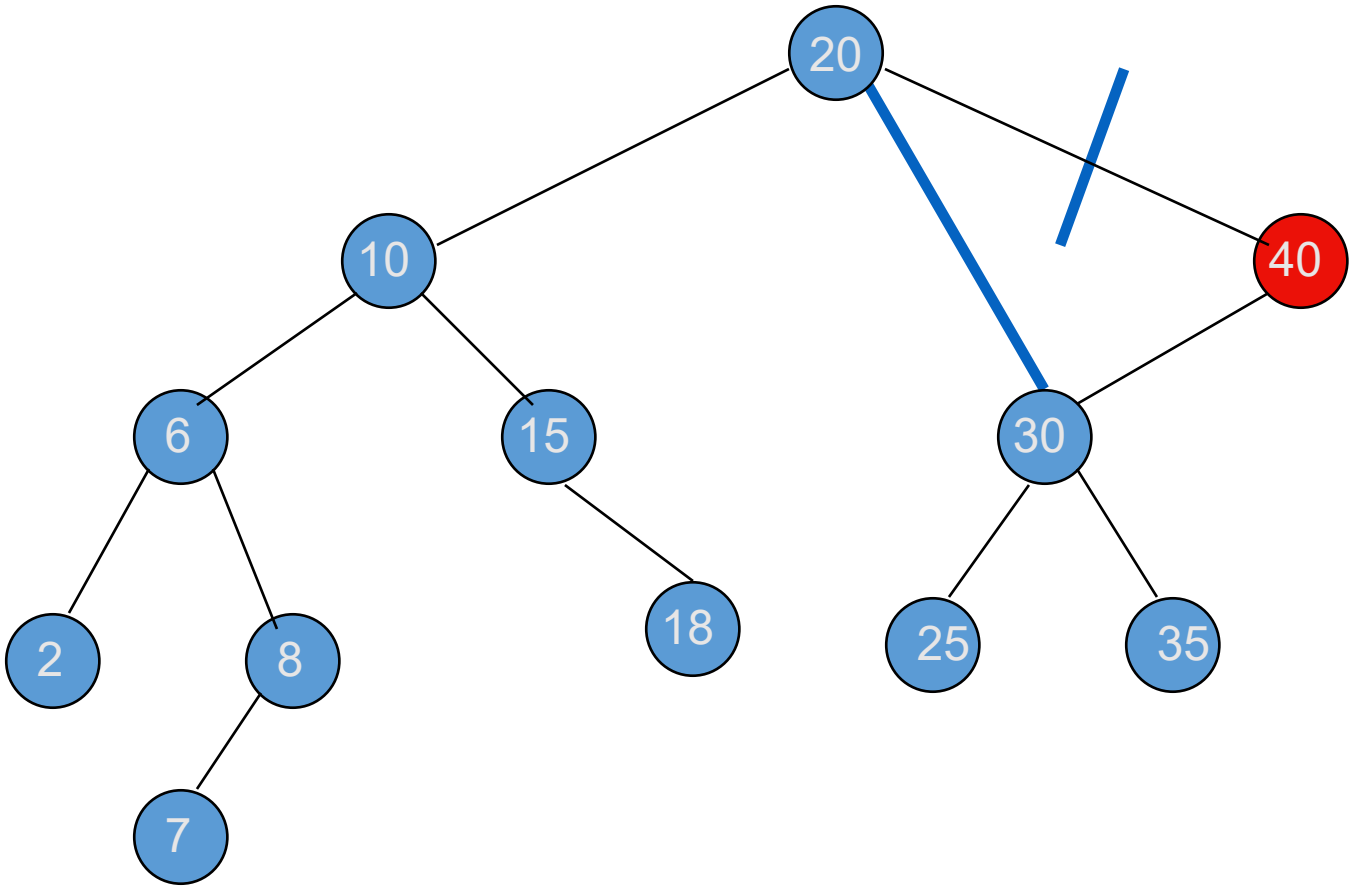
Remove a leaf element. key = 7

1. Penghapusan Node Daun (Node 35)



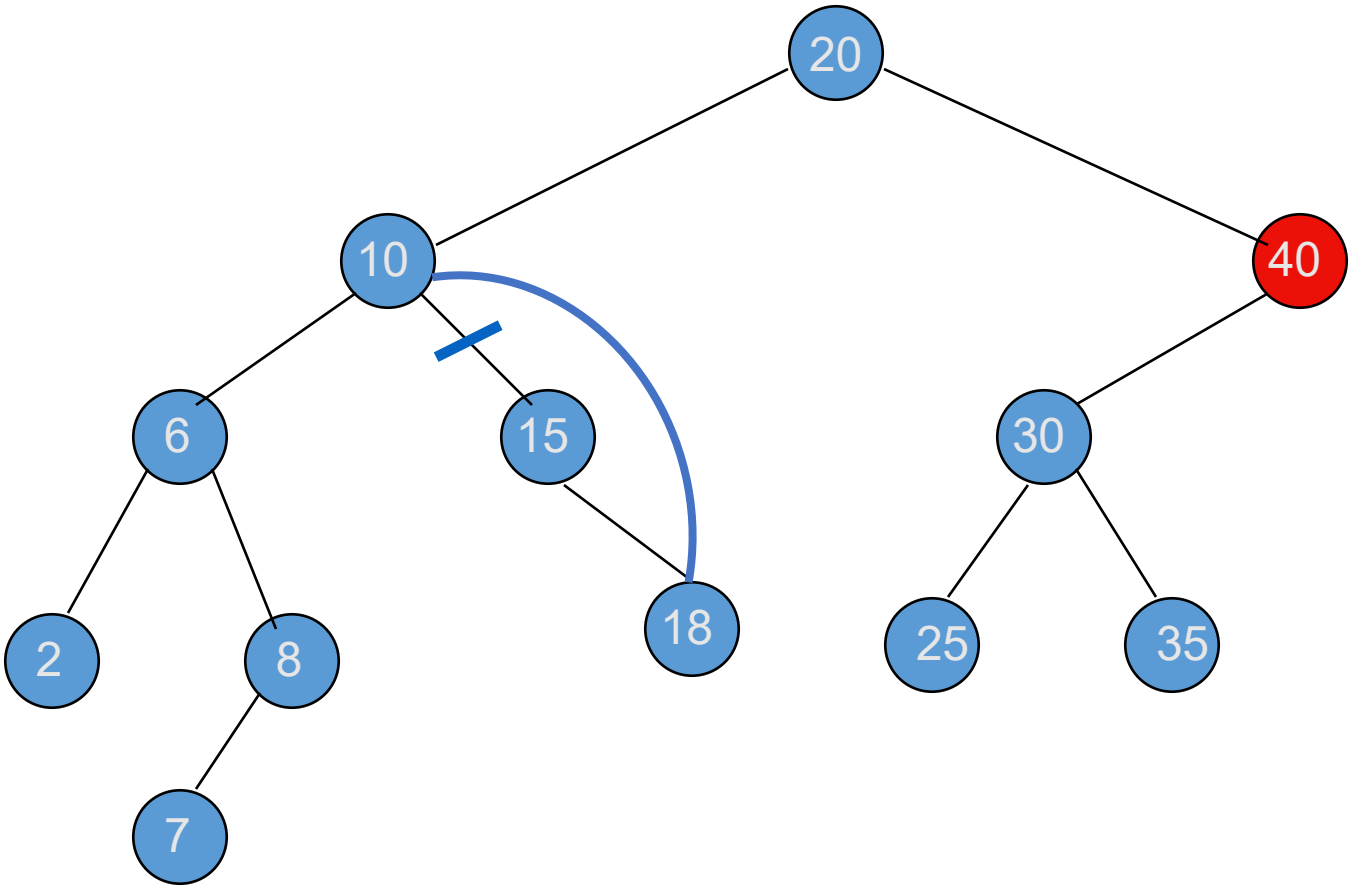
Remove a leaf element. key = 35

2. Penghapusan Node Ber-degree 1



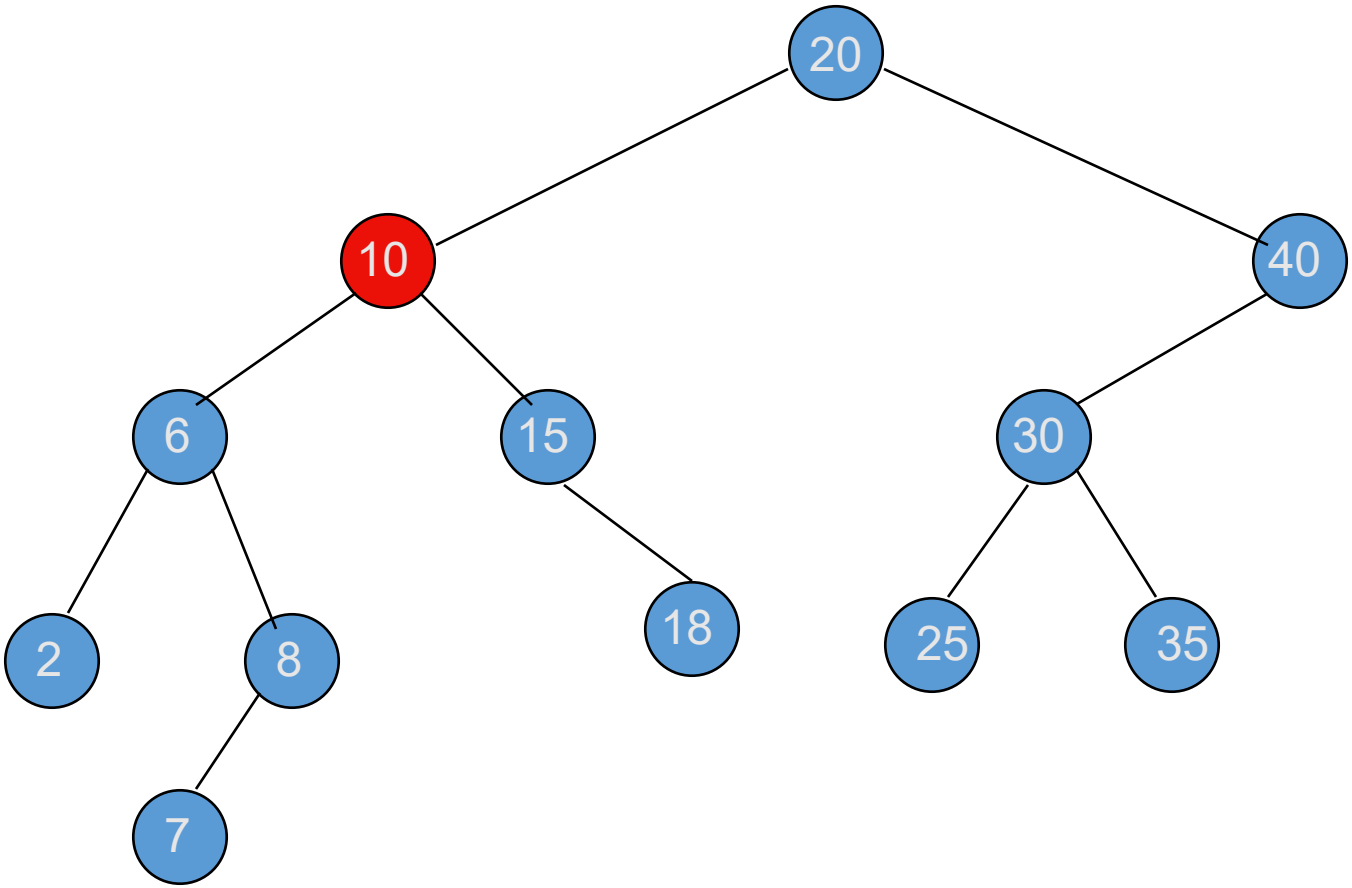
Remove from a degree 1 node. key = 40

2. Penghapusan Node Ber-degree 1



Remove from a degree 1 node. key = 15

3. Penghapusan Node Ber-degree 2



Remove from a degree 2 node. key = 10

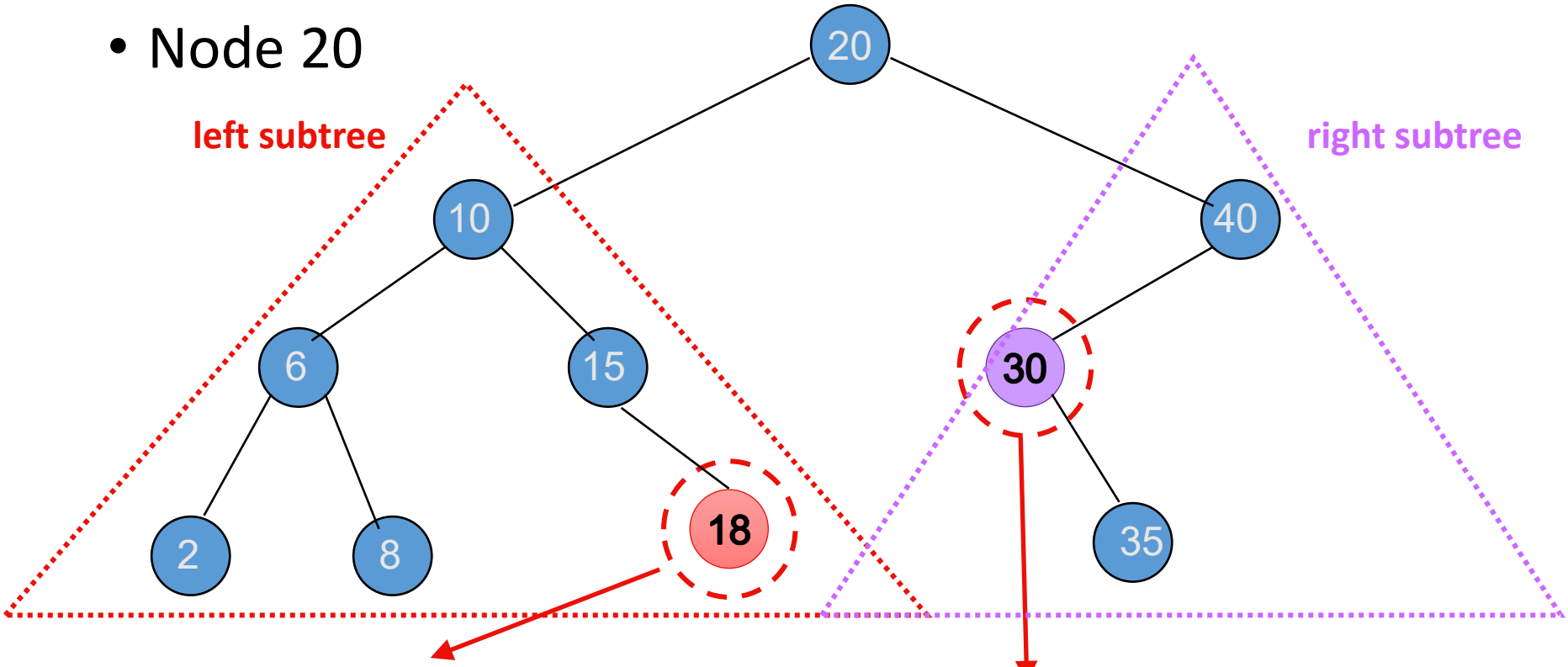
3. Penghapusan Node Ber-degree 2

- Penghapusan node ber-degree 2, akan digantikan oleh node:
- largest key di left subtree/*inorder predecesor*

  atau
- smallest key di right subtree/*inorder successor*

Contoh

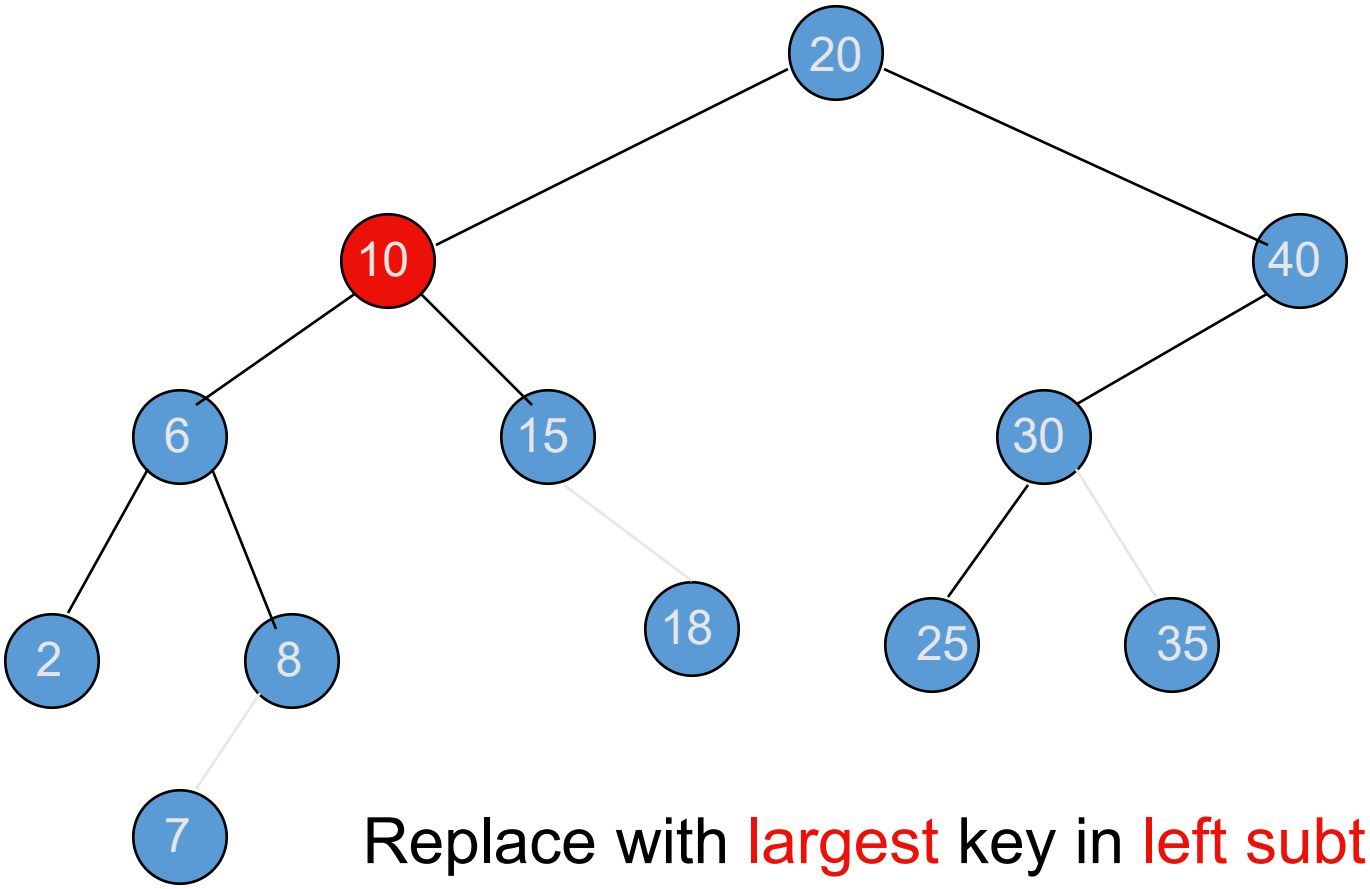
- Node 20



Predecessor node 25 adalah node paling kanan di subtree kiri, yaitu 18

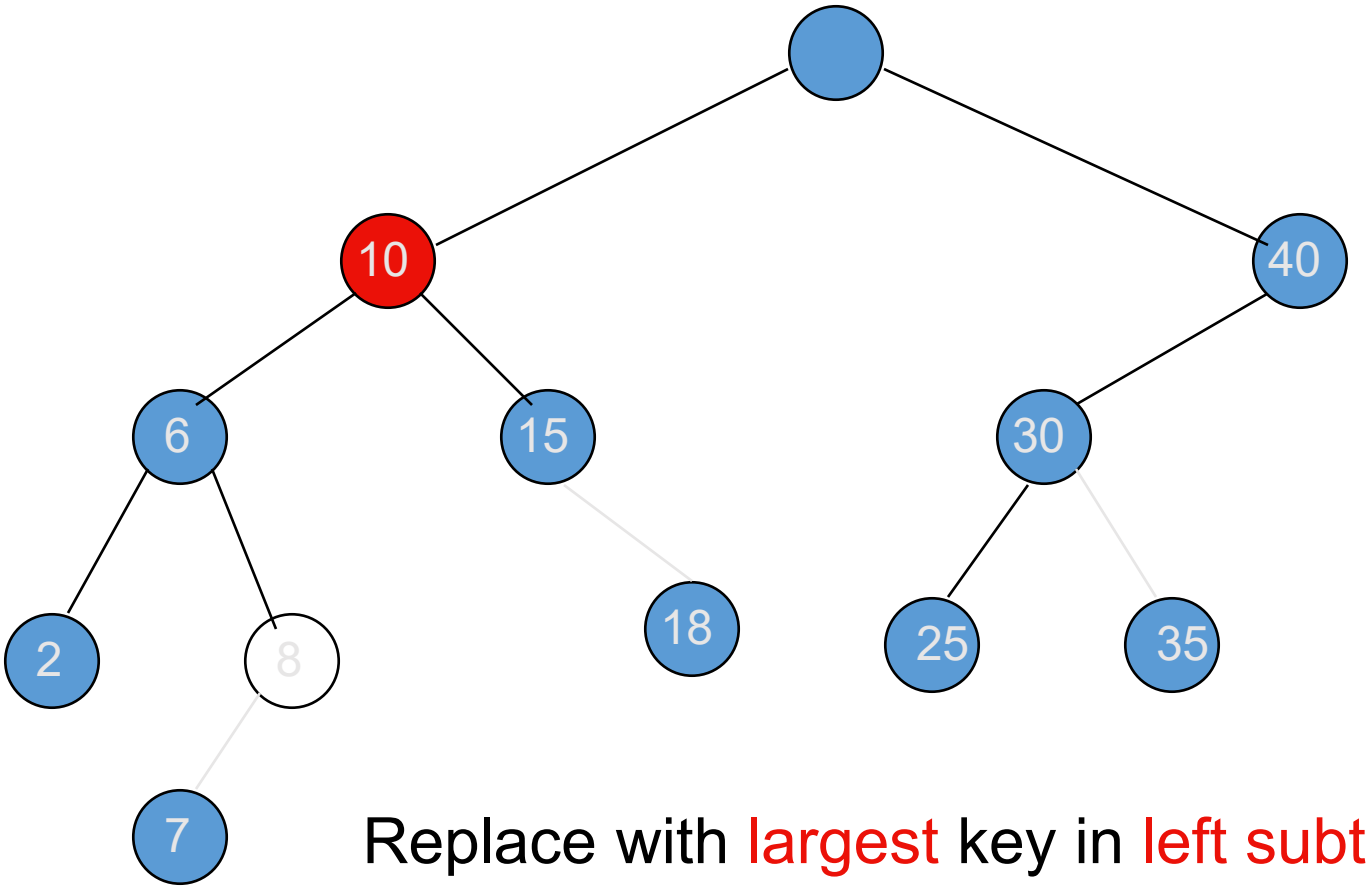
Successor node 25 adalah node paling kiri di subtree kanan, yaitu 30

3. Penghapusan Node Ber-degree 2



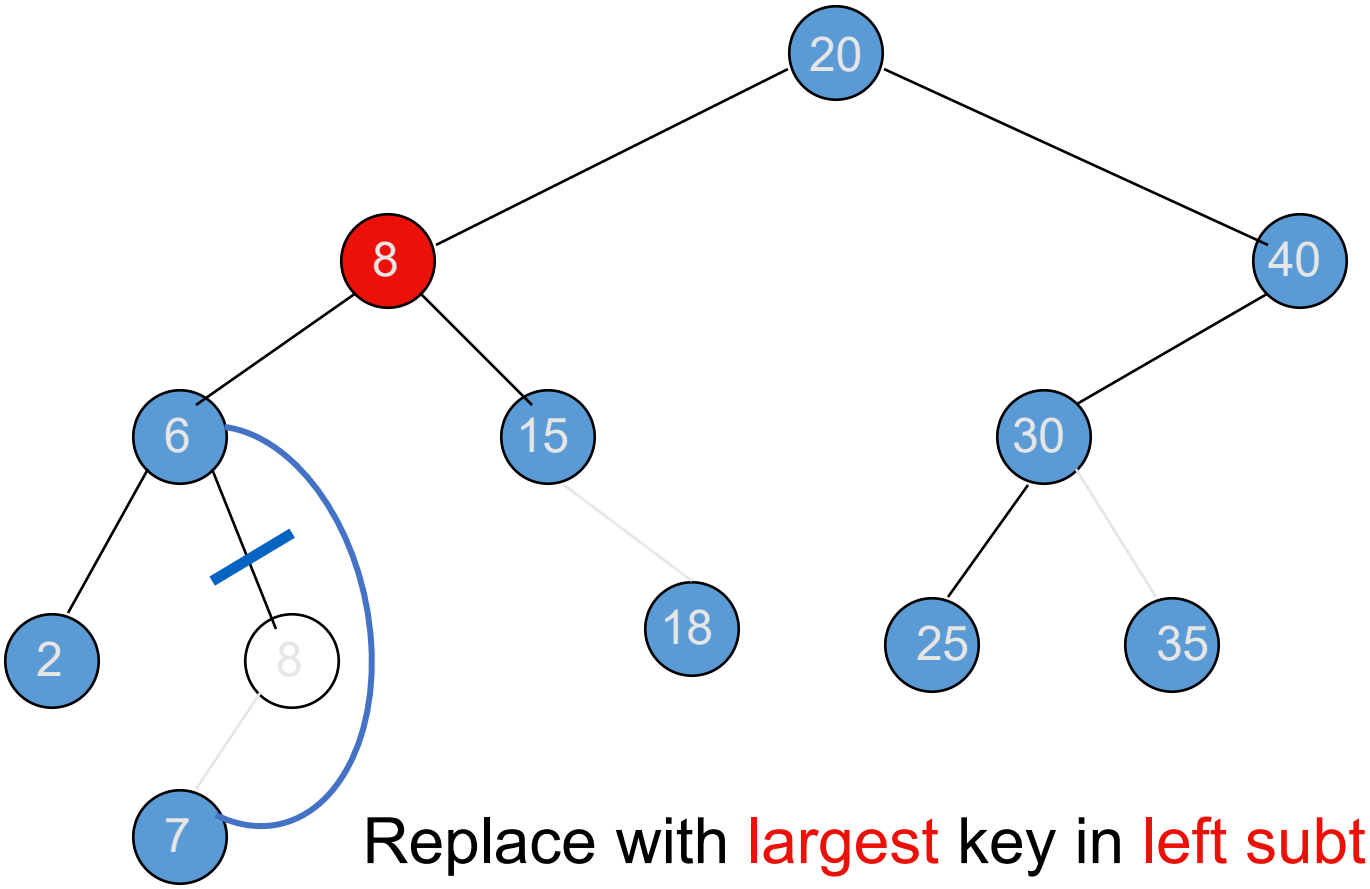
Replace with largest key in left subtree/*inorder predecesor* (or smallest in right subtree/*inorder successor*).

3. Penghapusan Node Ber-degree 2



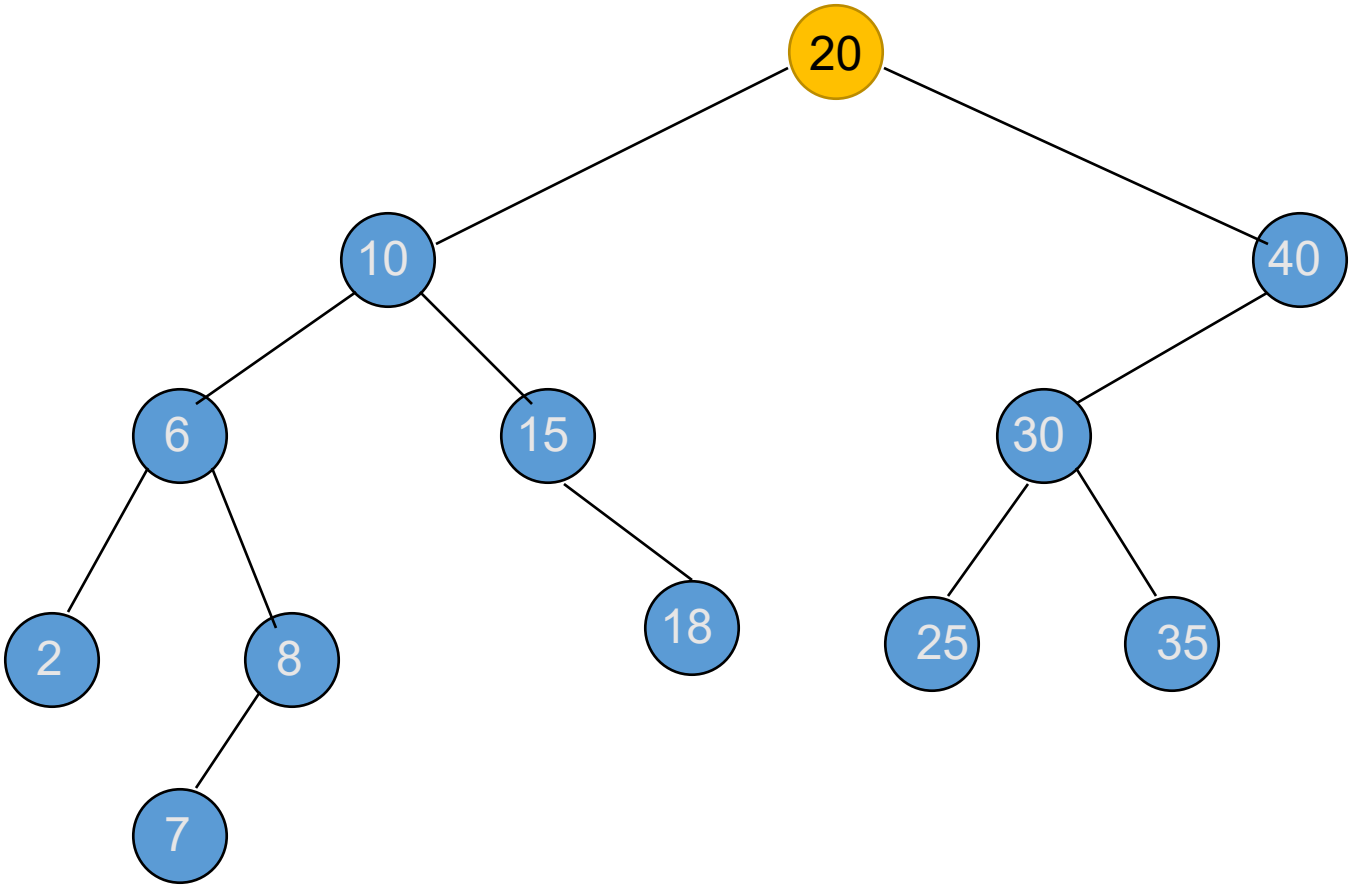
Replace with largest key in left subtree/*inorder predecesor* (or smallest in right subtree/*inorder successor*).

3. Penghapusan Node Ber-degree 2



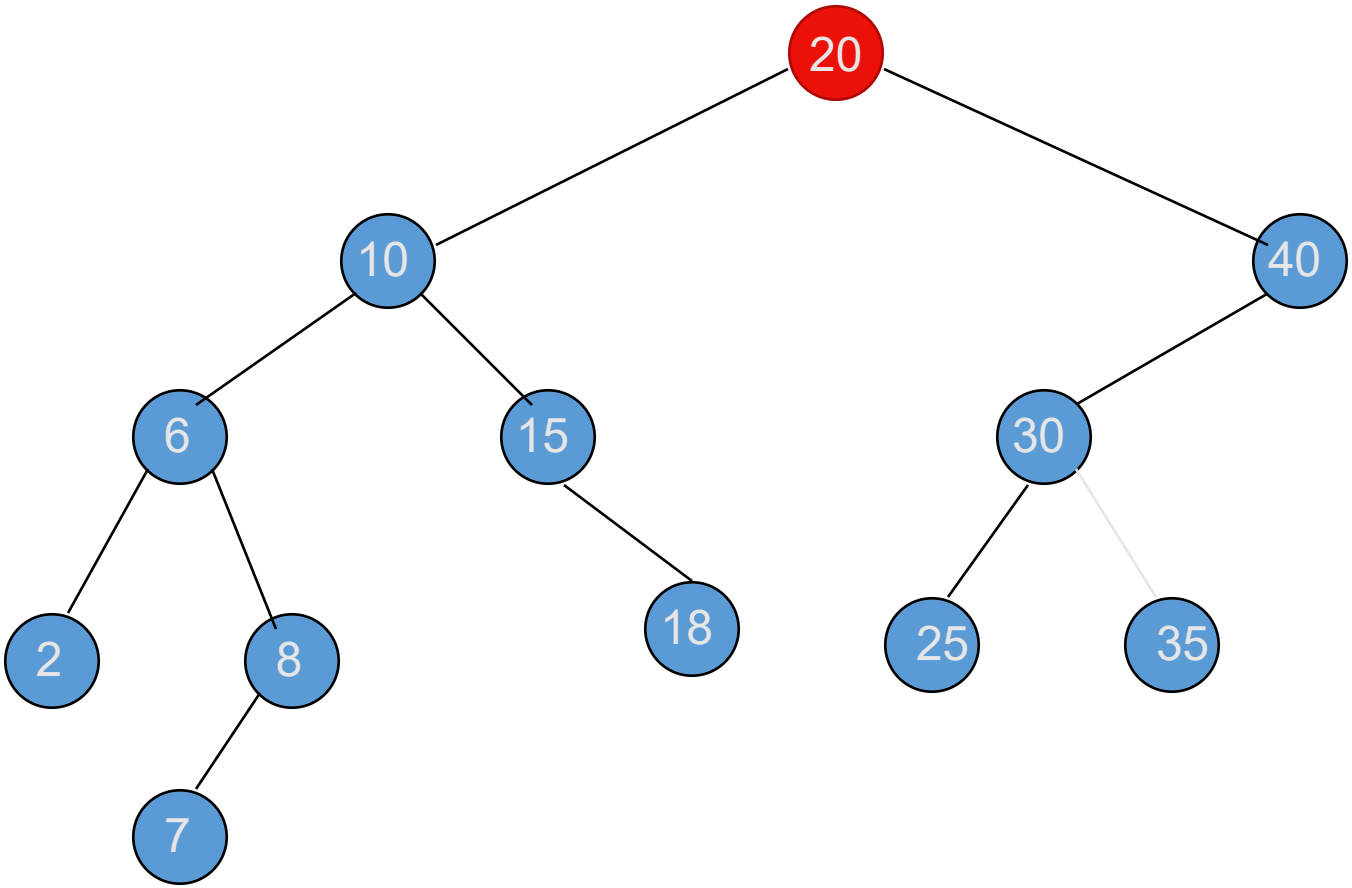
Replace with largest key in left subtree/*inorder predecesor* (or smallest in right subtree/*inorder successor*).

Latihan 1 – Inorder Successor



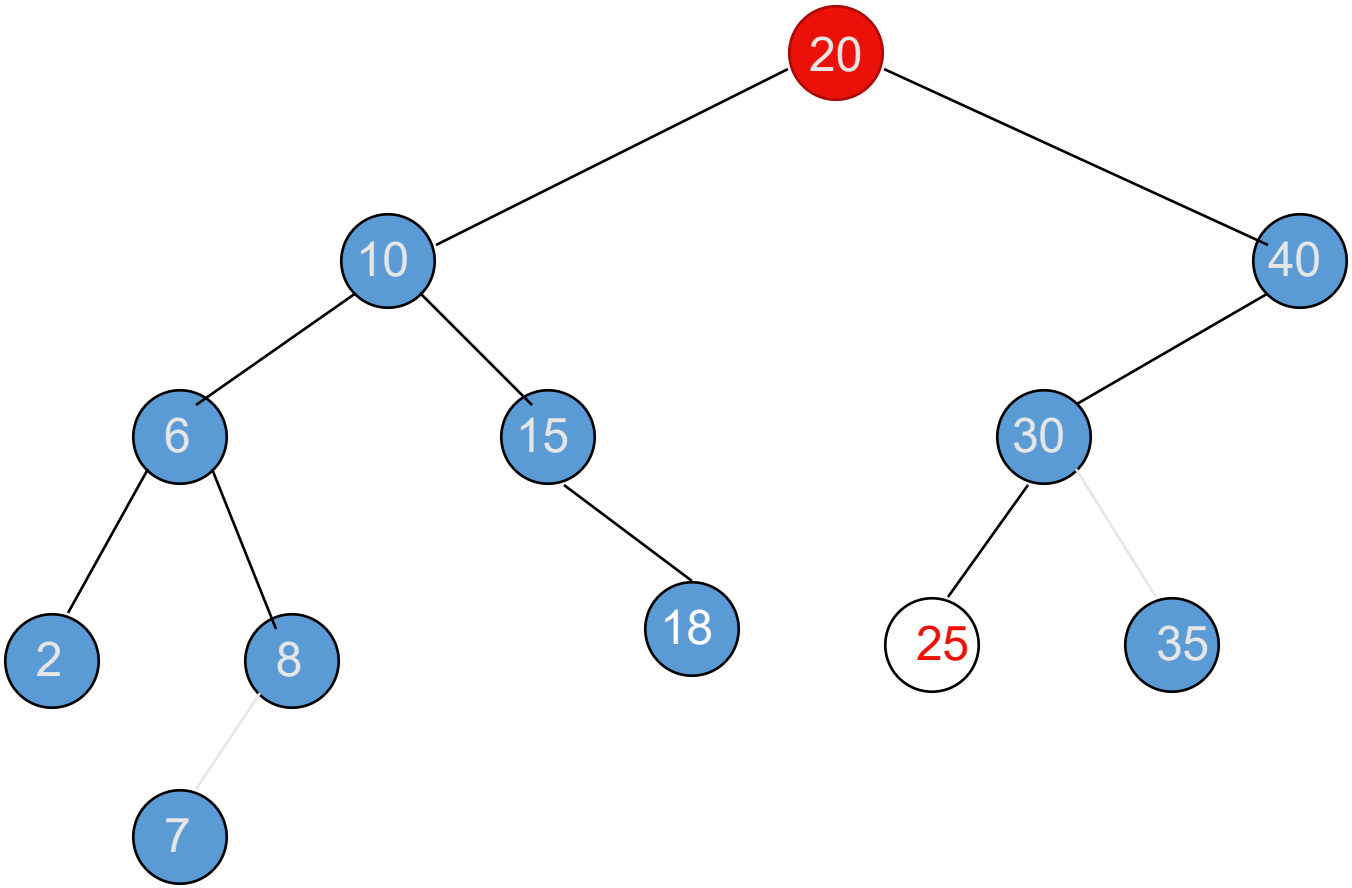
Remove from a degree 2 node. key = 20

Penghapusan Node Ber-degree 2



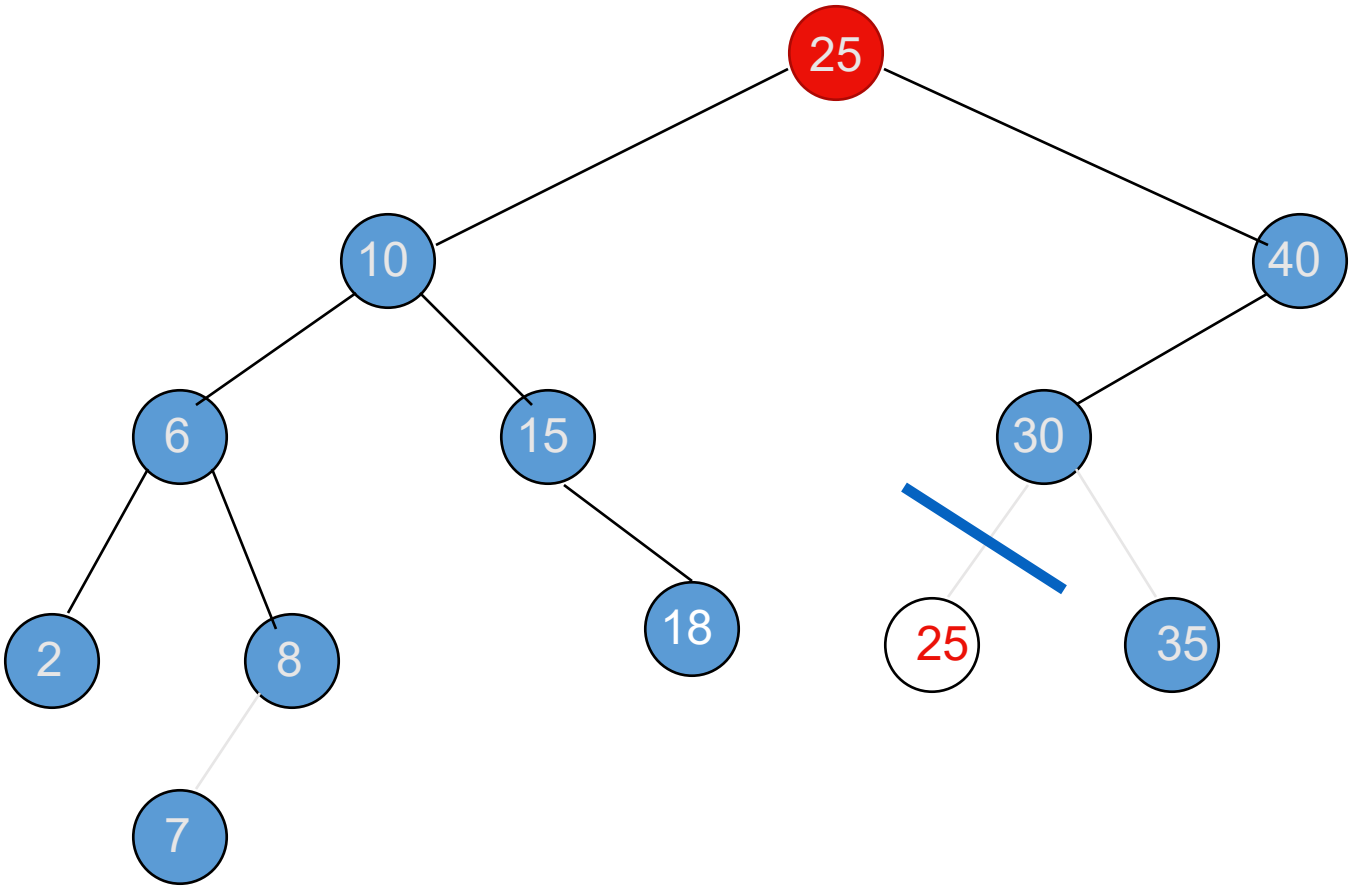
Replace with smallest in right subtree (*Inorder Successor*).

Penghapusan Node Ber-degree 2



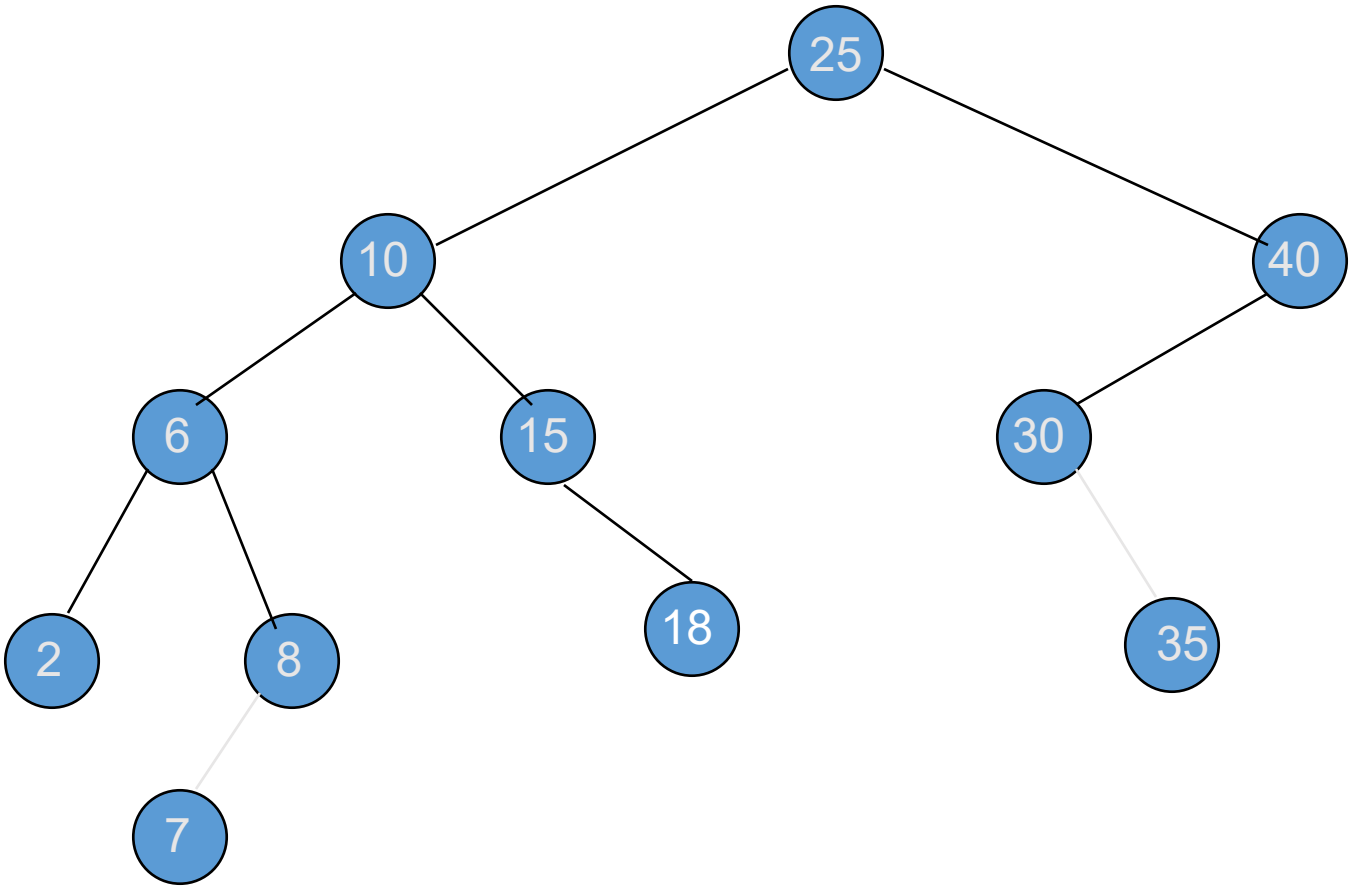
Replace with smallest in right subtree (*Inorder Successor*).

Penghapusan Node Ber-degree 2

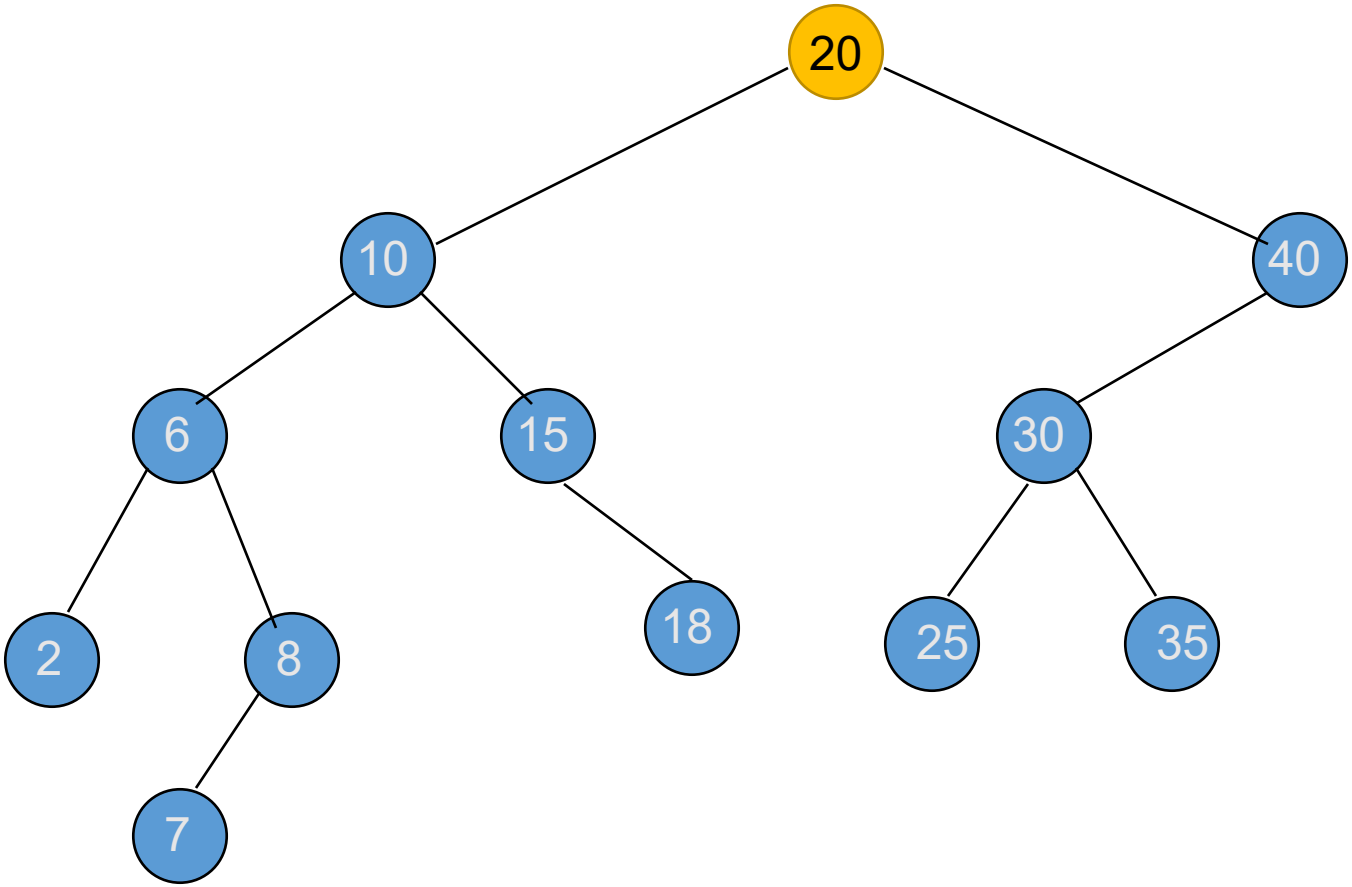


Replace with smallest in right subtree (*Inorder Successor*).

Hasil Akhir

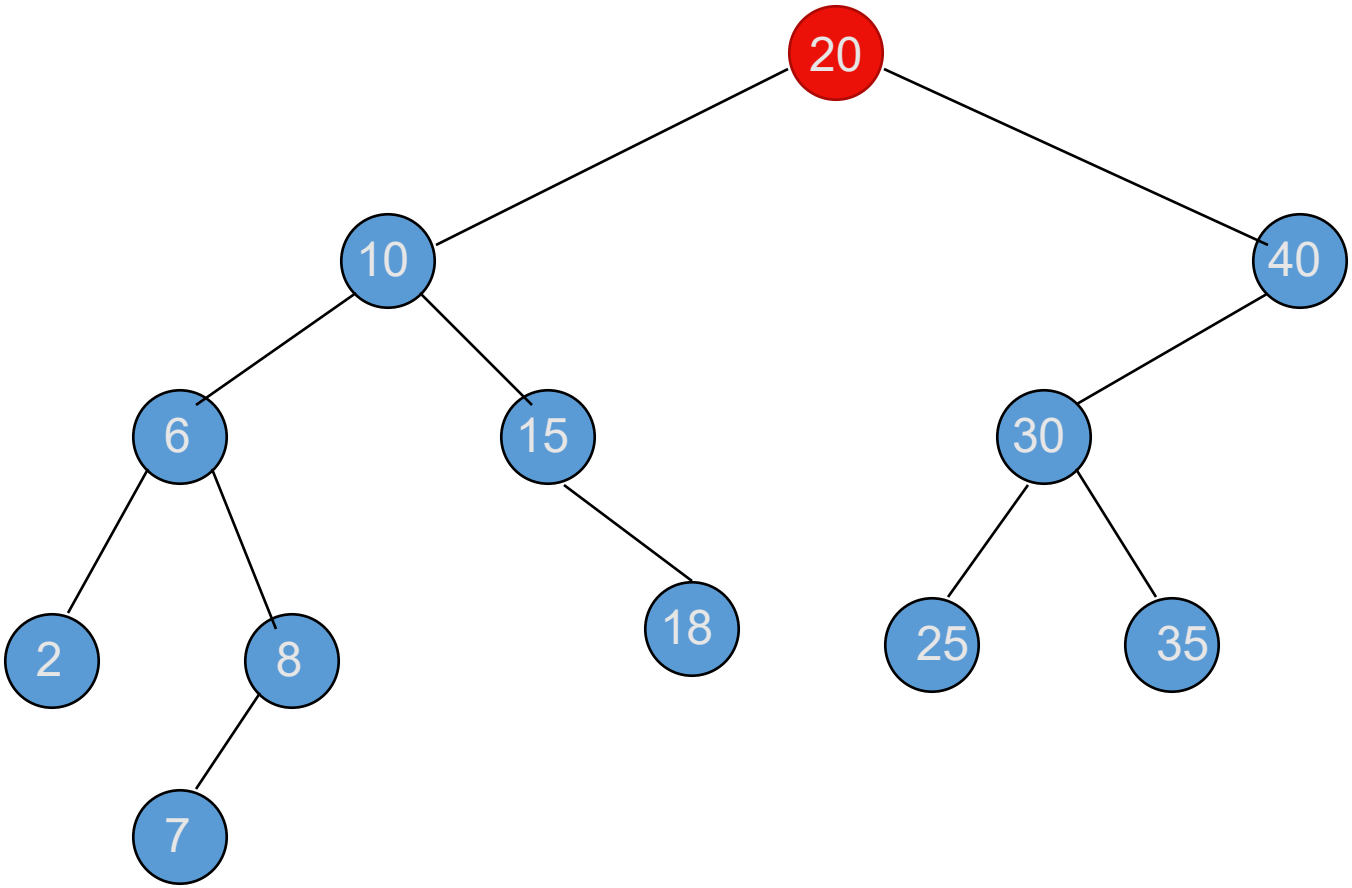


Latihan 2 – Inorder Predecessor



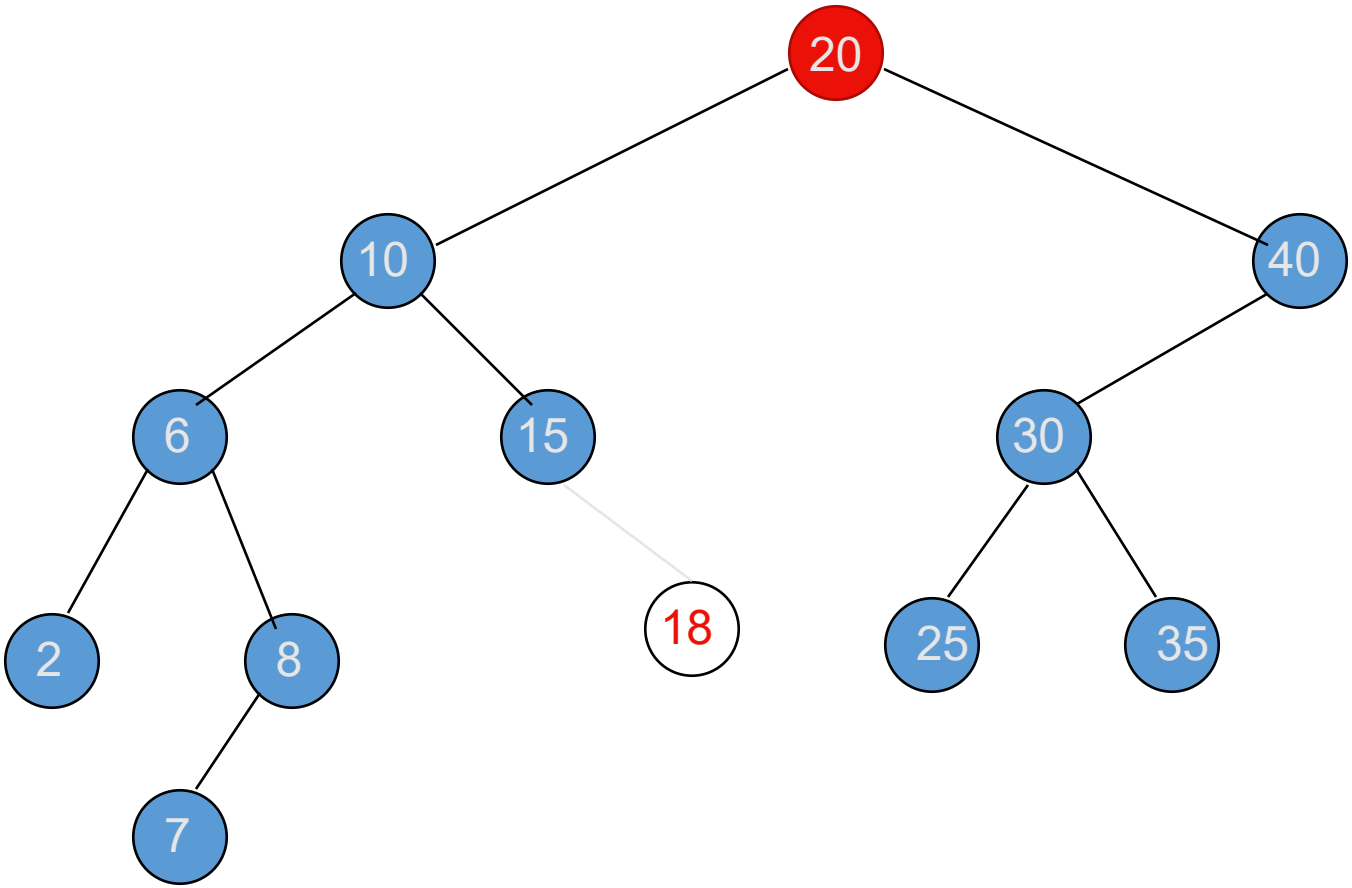
Remove from a degree 2 node. key = 20

Penghapusan Node Ber-degree 2



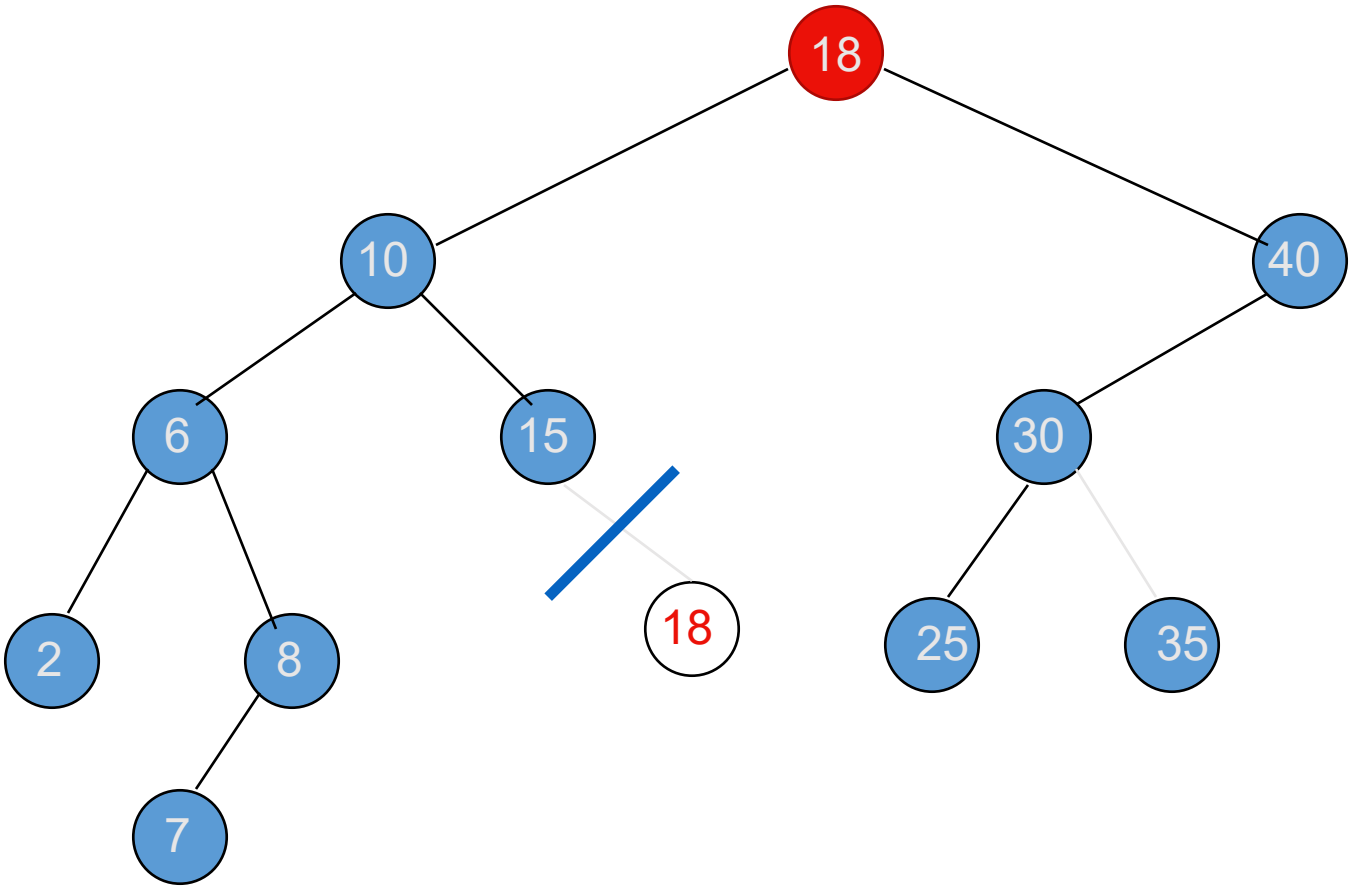
Replace with largest in left subtree (*Inorder Predecesor*).

Penghapusan Node Ber-degree 2



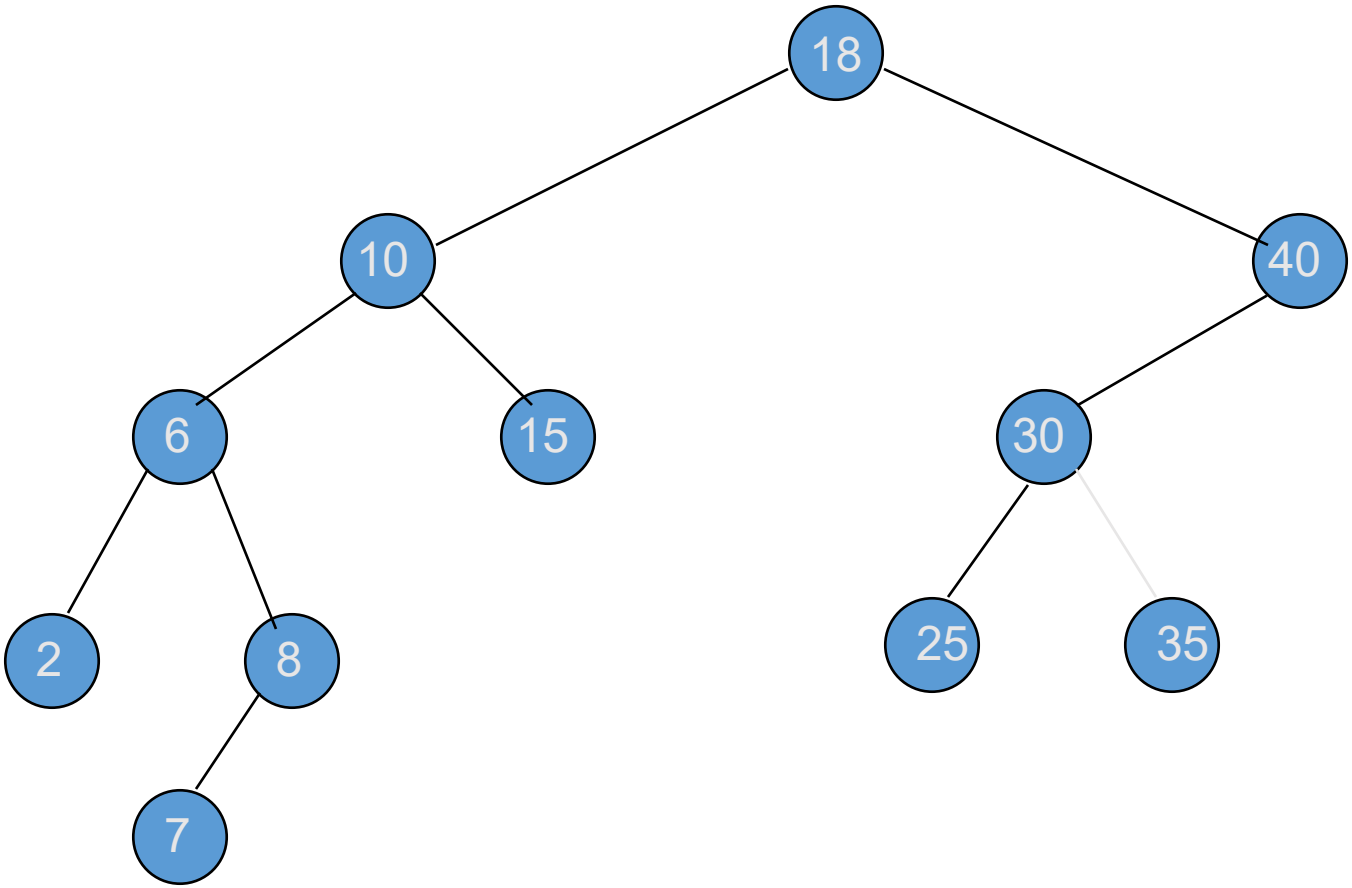
Replace with largest in left subtree (*Inorder Predecesor*).

Penghapusan Node Ber-degree 2



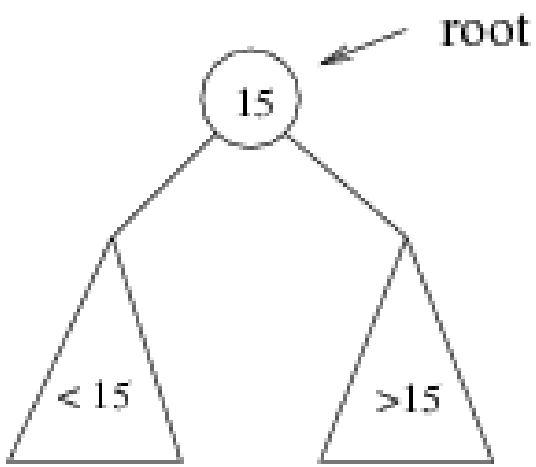
Replace with largest in left subtree (*Inorder Predecesor*).

Hasil Akhir



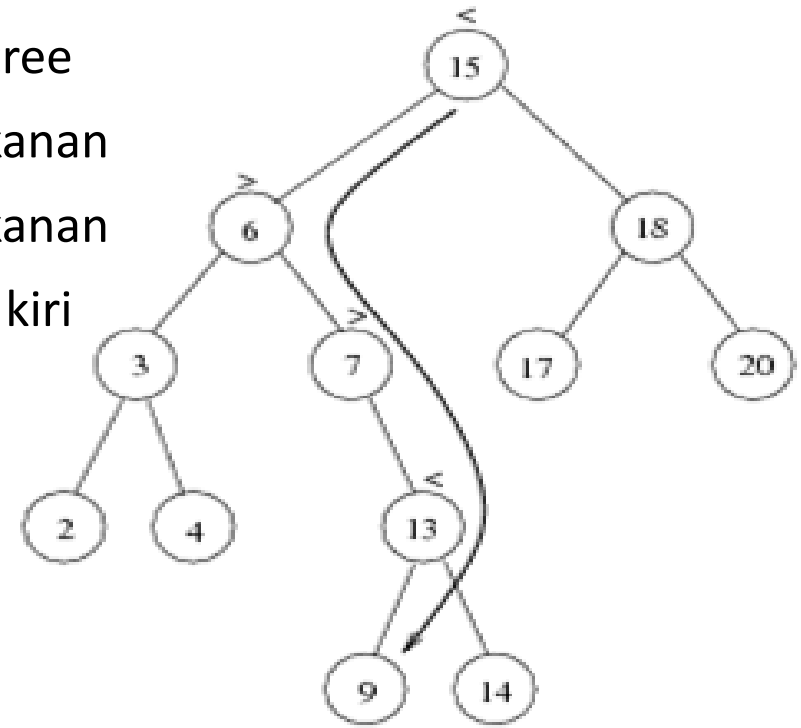
Pencarian pada BST

- Jika mencari elemen bernilai 15, maka akan langsung ditemukan.
- Jika mencari elemen bernilai < 15, maka kita cari di subpohon kiri.
- Jika mencari elemen bernilai > 15, maka kita cari di subpohon kanan.



Pencarian 9

1. Bandingkan 9:15, pergi ke kiri subtree
2. Bandingkan 9:6, pergi ke subtree kanan
3. Bandingkan 9:7, pergi ke subtree kanan
4. Bandingkan 9:13, pergi ke subtree kiri
5. Bandingkan 9:9, ditemukan!



findMin/ findMax

- `findMin`: mengembalikan node dengan elemen terkecil pada BST (akan digunakan untuk *inorder successor*)
- Pencarian dimulai dari root dan bergerak ke kiri terus sepanjang subtree kiri dan berhenti pada elemen terakhir.

- Proses serupa terhadap metode `findMax` (digunakan untuk *inorder predecessor*)