Menghapus Simpul pada *Binary Tree*

(TIB11 – Struktur Data)

Pertemuan 21, 22

Sub-CPMK

- Mahasiswa mampu melakukan penghapusan simpul binary tree (C3, A3)

Materi:

1. Hapus *Leaf*
2. Hapus Simpul dengan Satu Anak
3. *Delete By Merging*
4. *Delete By Copying*

1. Hapus *Leaf*

1.1. Menghapus Simpul

Rules:

Setiap *key* pada *subtree* sebelah kanan harus lebih besar dari pada *key-key* pada *subtree* sebelah kiri

1.2. Kondisi Penghapusan Simpul

Penghapusan tergantung pada kondisi

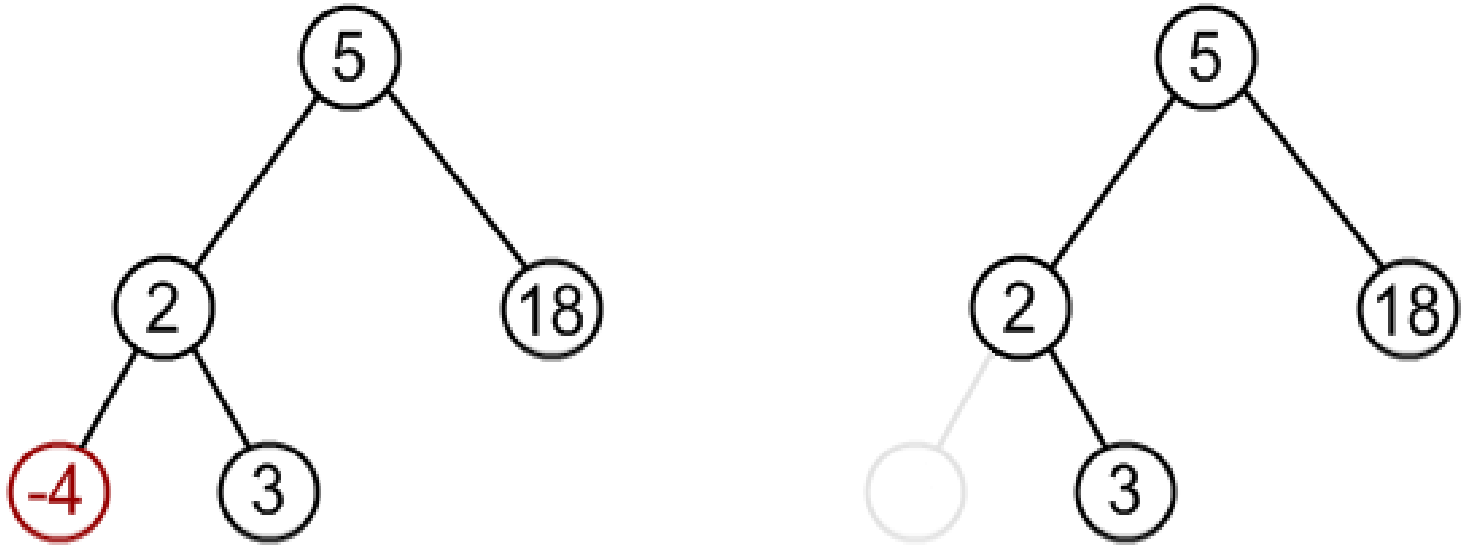
- Pada *Leaf* → hapus saja tanpa penangan masalah
- Pada node dengan satu *child* → *child* satu-satunya tersebut akan menjadi *child* dari *grandparent* menggantikan *node* yang dihapus
- Pada *Node* dengan dua *child* dapat dilakukan dengan cara:
  - *Deletion by merging*
  - *Deletion by copying*

Gambar dapat dilihat pada masing-masing pembahasan di slide berikutnya

1.3. Hapus Pada *Leaf*

- Penghapusan simpul pada *leaf* dapat dilakukan tanpa perlakuan khusus apapun, karena sebagai *leaf*, suatu simpul tidak memiliki keturunan yang harus diatur agar memenuhi persyaratan node sebelah kiri harus lebih kecil daripada *node* sebelah kanan

1.3. Hapus Pada *Leaf* (Lanj.)



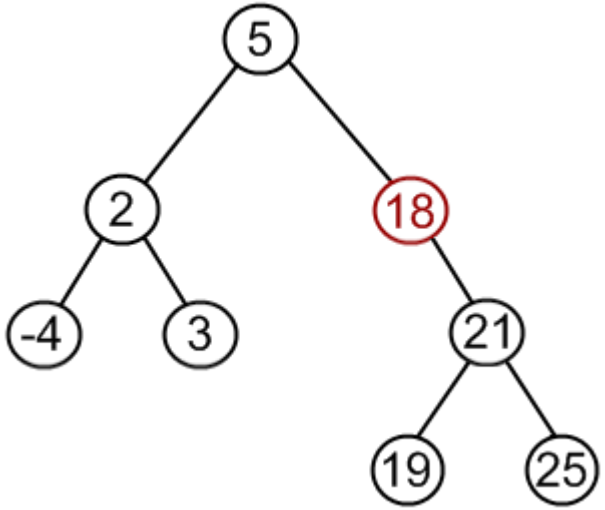
2. Hapus Simpul Dengan Satu Anak

Hapus simpul dengan satu anak

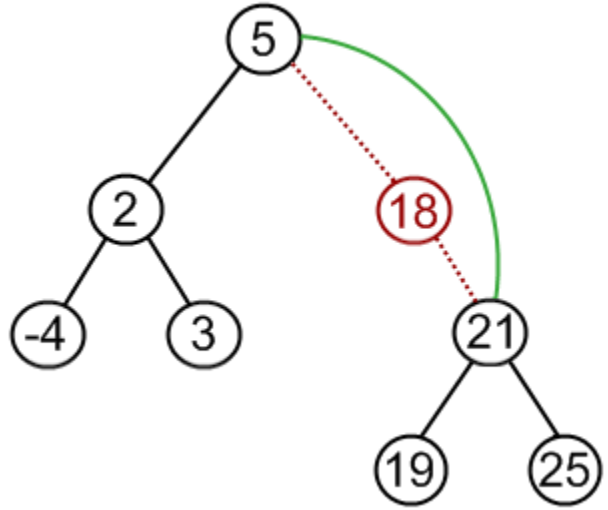
- Node dengan satu anak hanya memiliki masalah menjadi anak dari induk manakah node dari anak yang dihapus tersebut
- Sedangkan anak dari anak node yang dihapus akan mengikuti anak dari node yang dihapus
- Maka perlakuan penghapusan node dengan satu anak dilakukan dengan anak satu-satunya tersebut akan menjadi anak dari *ancestor*/leluhur node yang dihapus

Hapus simpul dengan satu anak (Lanj.)

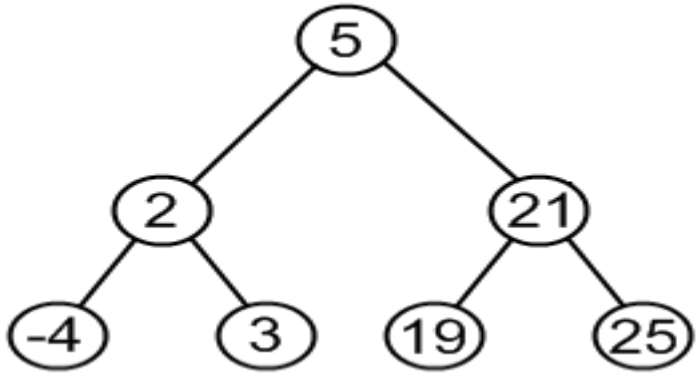
- Misal akan menghapus node 18



- Maka node 21 akan menjadi anak dari node 5 menggantikan node 18



- Hasil akhir



3. *Delete by Merging*

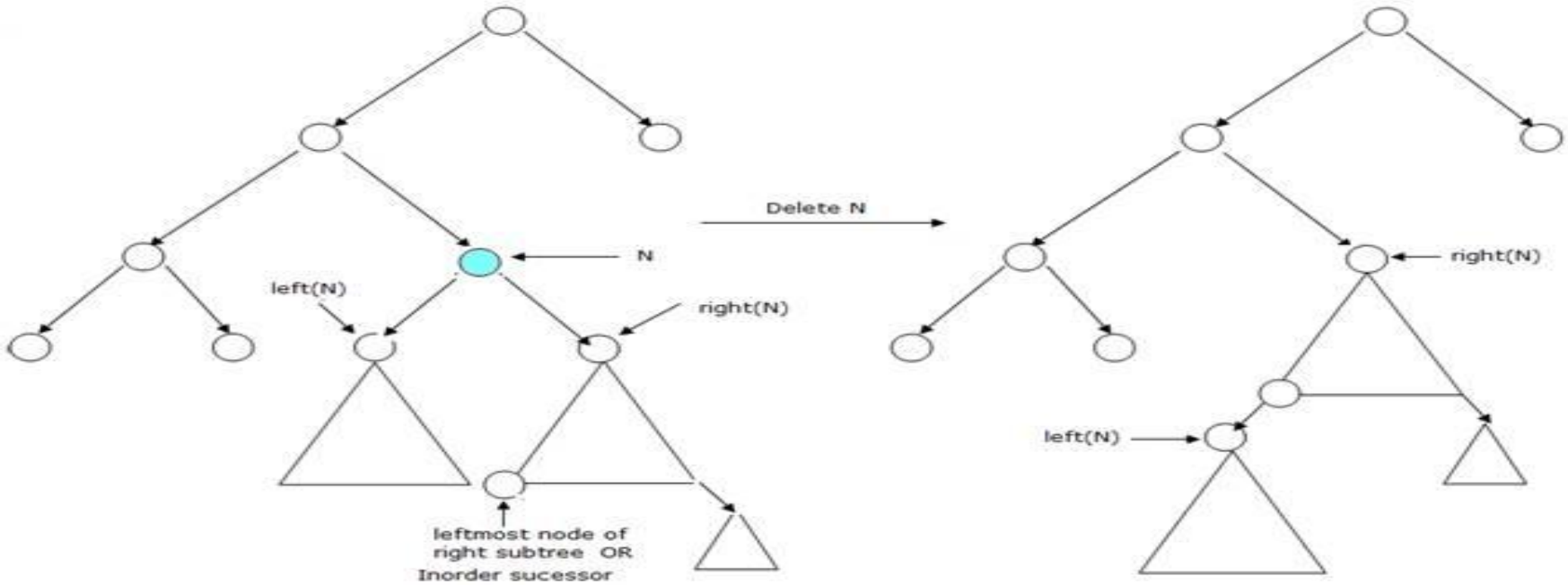
- *replace* ***right node*** sebagai *node* induk dan gabungkan *node* kiri ke *leftmost node* dari *subtree* kanan

  Atau lakukan kebalikannya

- *replace* ***left node*** sebagai *node* induk dan gabungkan *node* kanan ke *rightmost node* dari *subtree* kiri

Lanj.

Deleting node N dengan dua *children by merging right subtree into left subtree*
Node yang ditunjuk oleh N akan dihapus,
Maka sub –tree yang ada di kanan node N akan menjadi pewaris menggantikan N
Sedangkan sub tree di kiri Node N akan menjadi anak node terkiri dari sub tree yang semula di kanan Node N



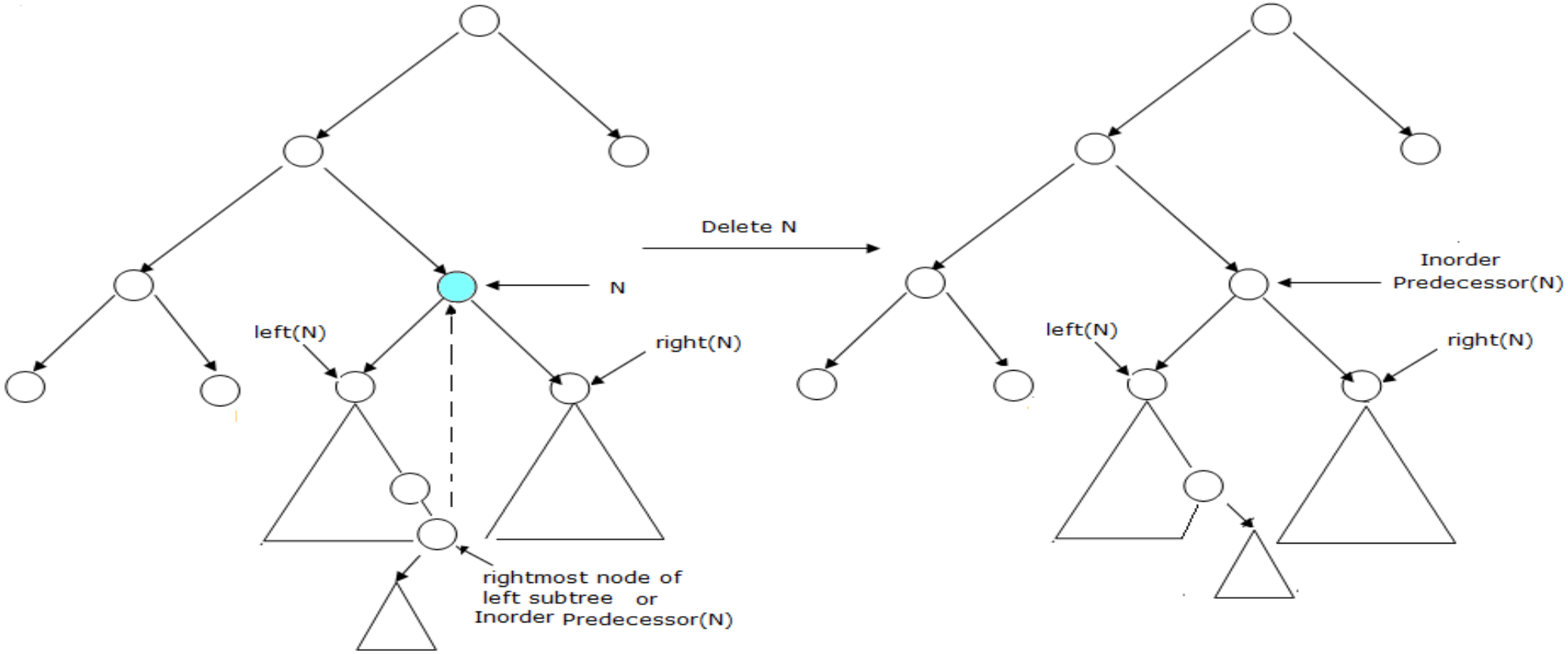
Sumber gambar: http://vle.du.ac.in/mod/book/view.php?id=5726&chapterid=3011

4. *Delete by Copying*

- Ganti ***key*** yang akan dihapus dengan *immediate predecessor* atau *immediate successor* (bisa juga dengan cara sebaliknya) dg cara:
  - Ambil *Subtree* sebelah kanan dari *node* yang akan dihapus, cari *leaf* yang paling kiri dari *node* yang harusnya dikunjungi secara *in-order*
  - Copykan *leaf* ke posisi *node* yang akan dihapus
  - Hapus *Leaf* yang sudah dicopykan menjadi node pengganti yang dihapus tadi

Lanj.

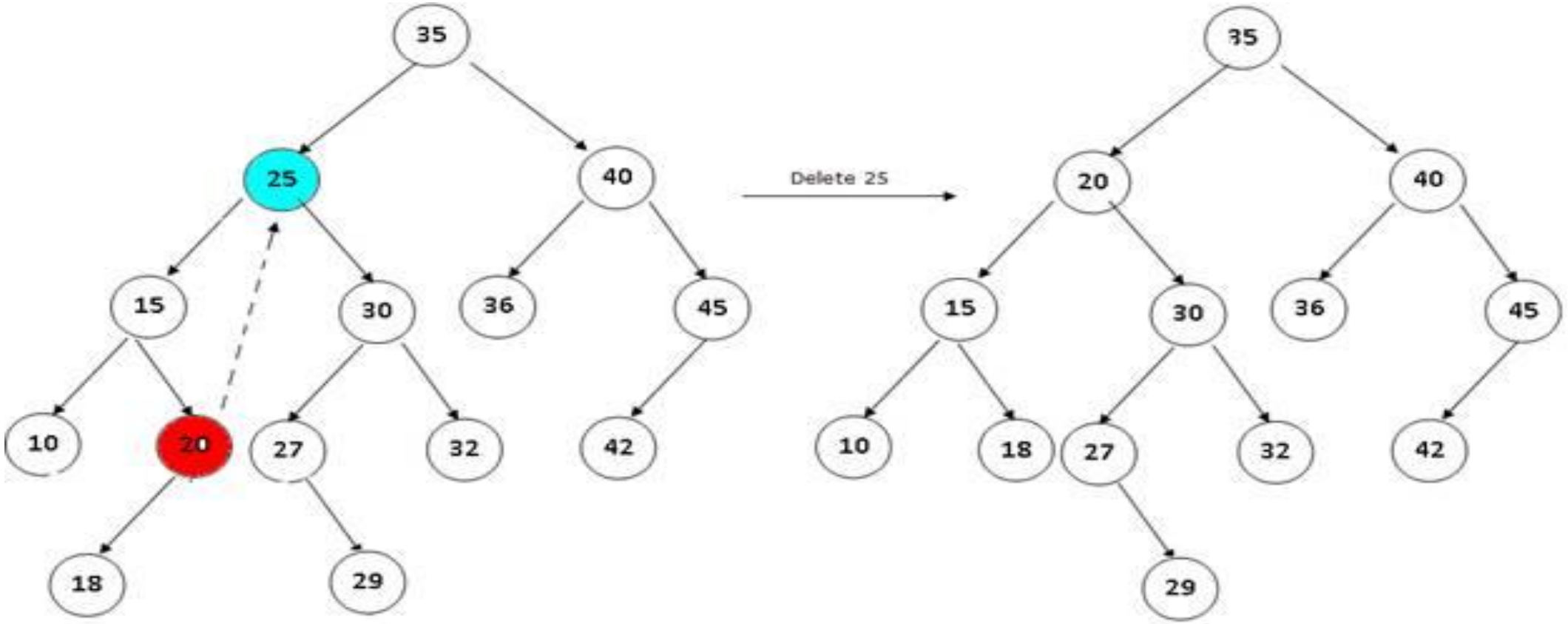
Deleting* N *by copying In-order Predecessor



Sumber gambar: http://vle.du.ac.in/mod/book/view.php?id=5726&chapterid=3023

Lanj.

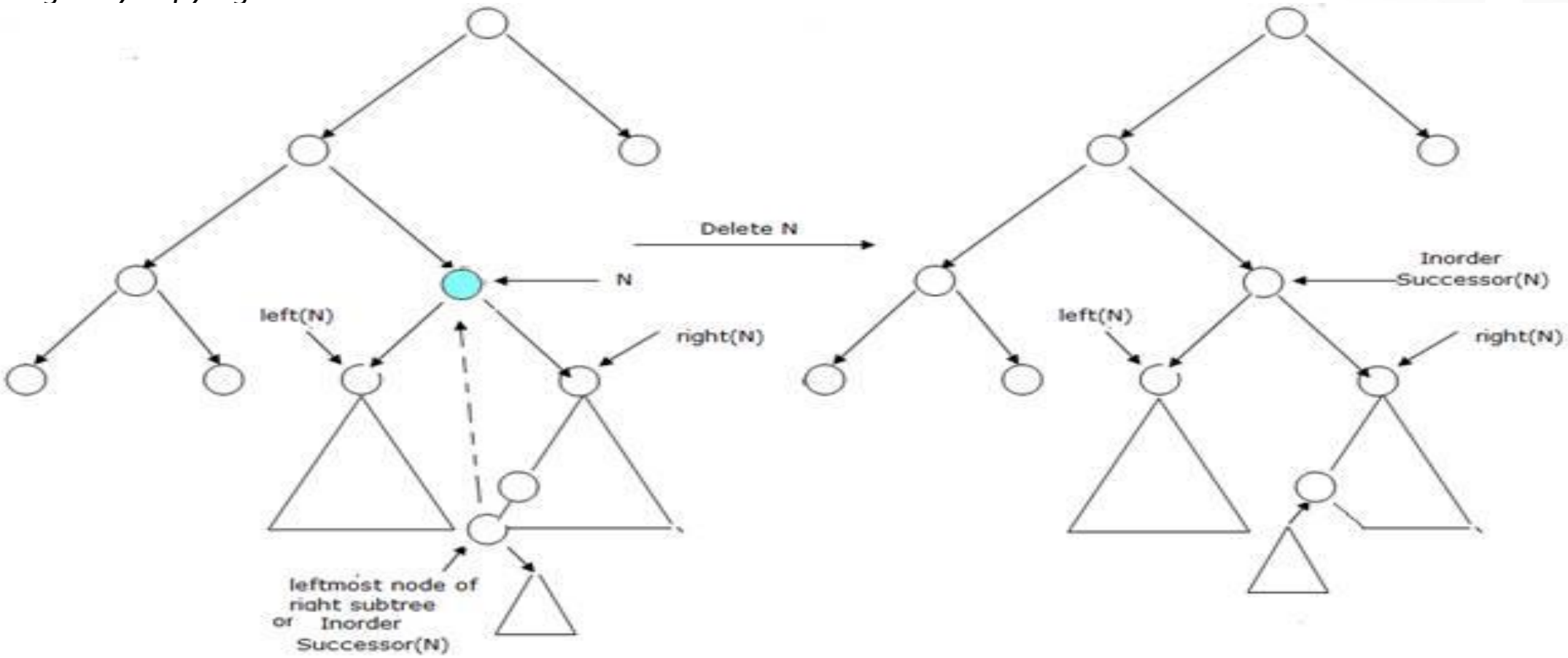
Contoh Hapus 25 dg *Deletion by copying* menggunakan *right-most inorder predecessor*



Sumber gambar: http://vle.du.ac.in/mod/book/view.php?id=5726&chapterid=3023

Lanj.

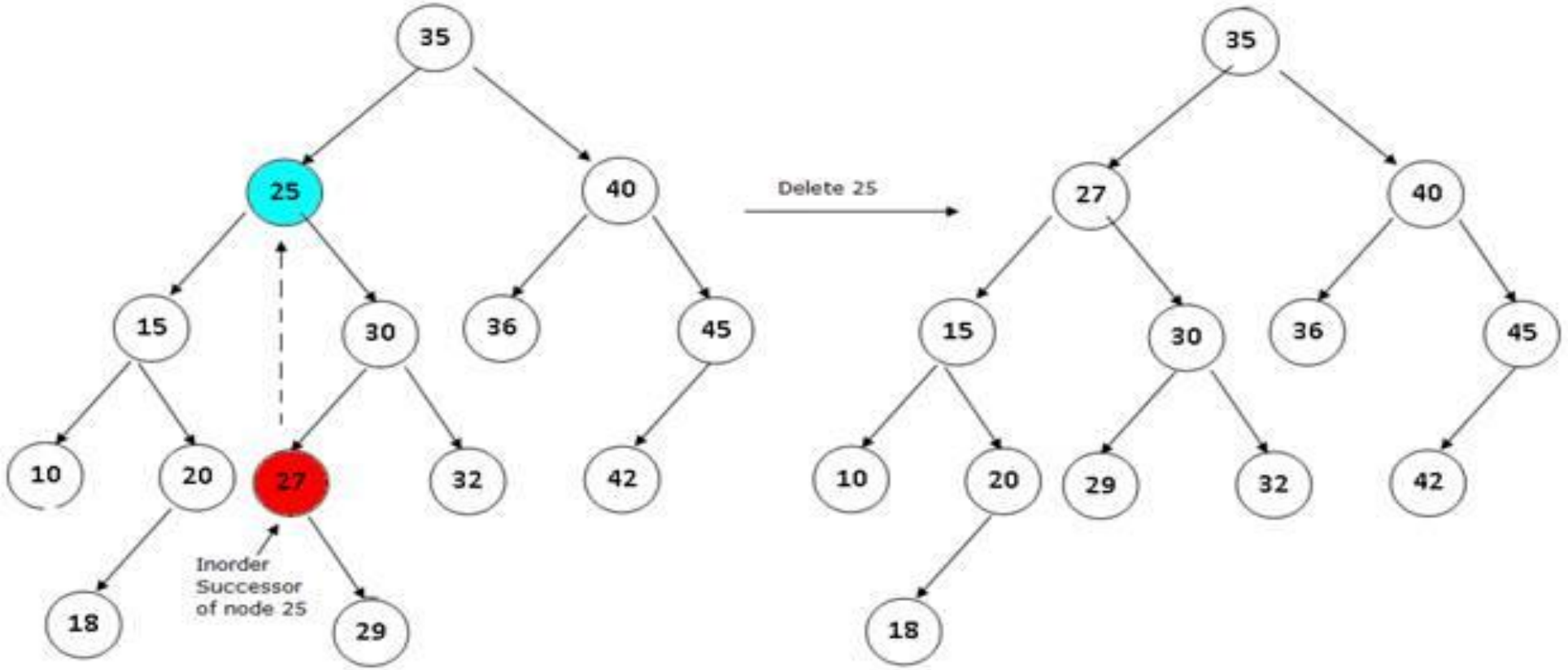
Deleting N *by copying Inorder Successor*



Sumber gambar: http://vle.du.ac.in/mod/book/view.php?id=5726&chapterid=3023

Lanj.

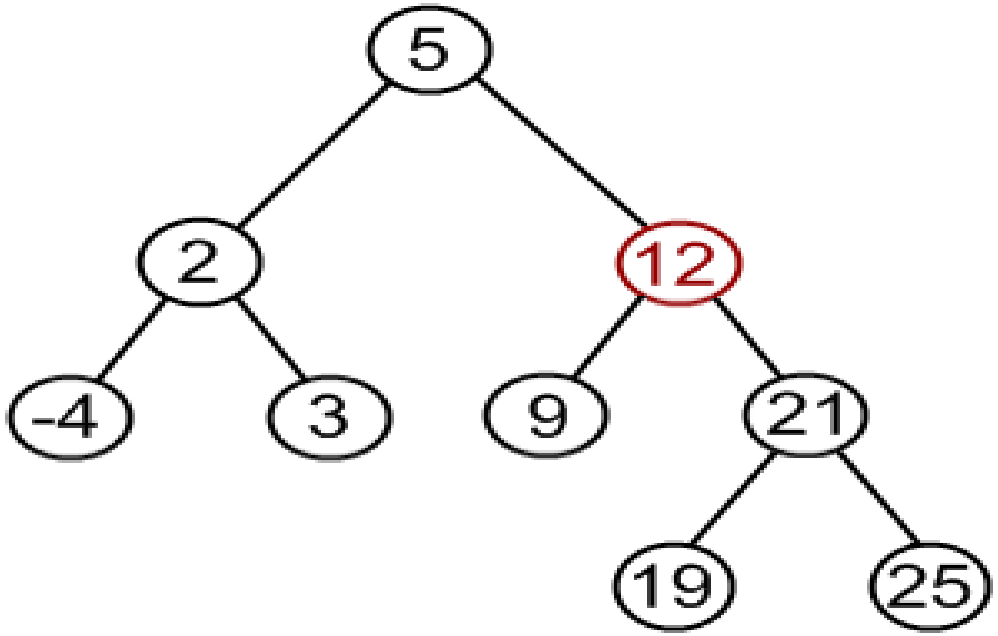
Contoh *Deleting* 25 dg *Deletion by copying* menggunakan *left-most inorder succesor*



Sumber gambar: http://vle.du.ac.in/mod/book/view.php?id=5726&chapterid=3023

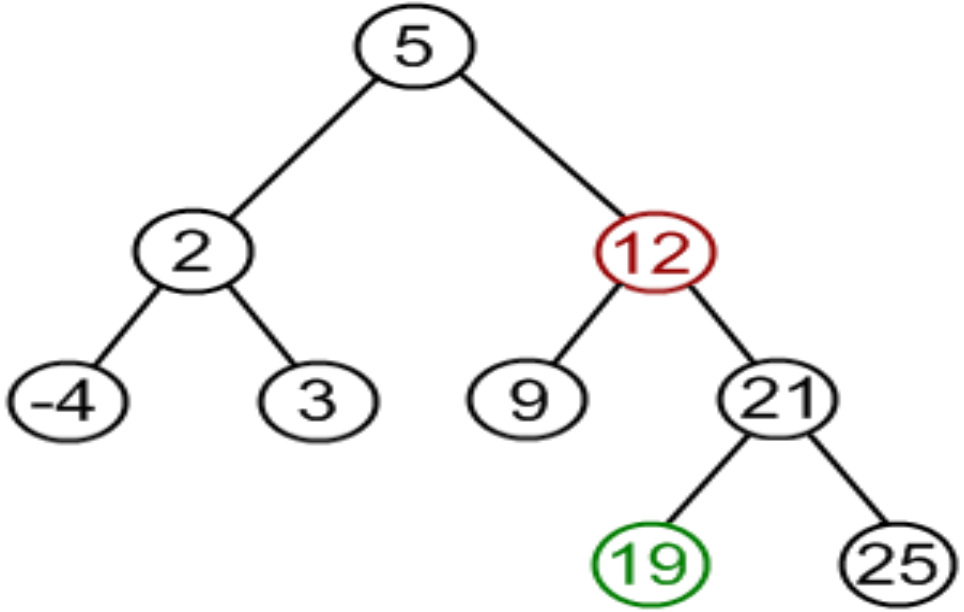
Lanj.

Contoh lain: Misalkan Akan hapus *node* 12



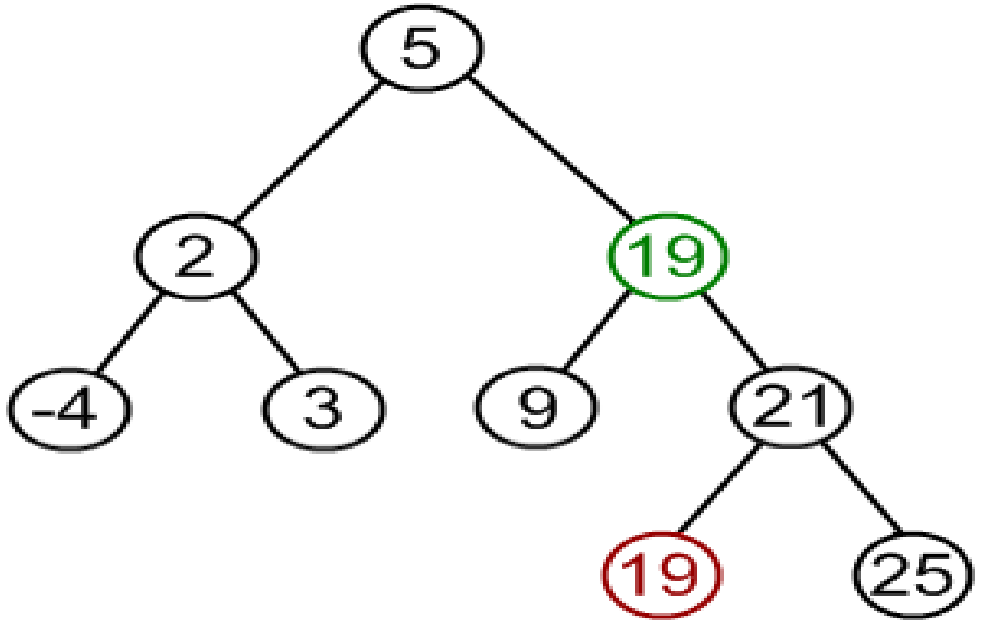
Lanj.

Node Subtree Kanan yang paling kiri adalah 19



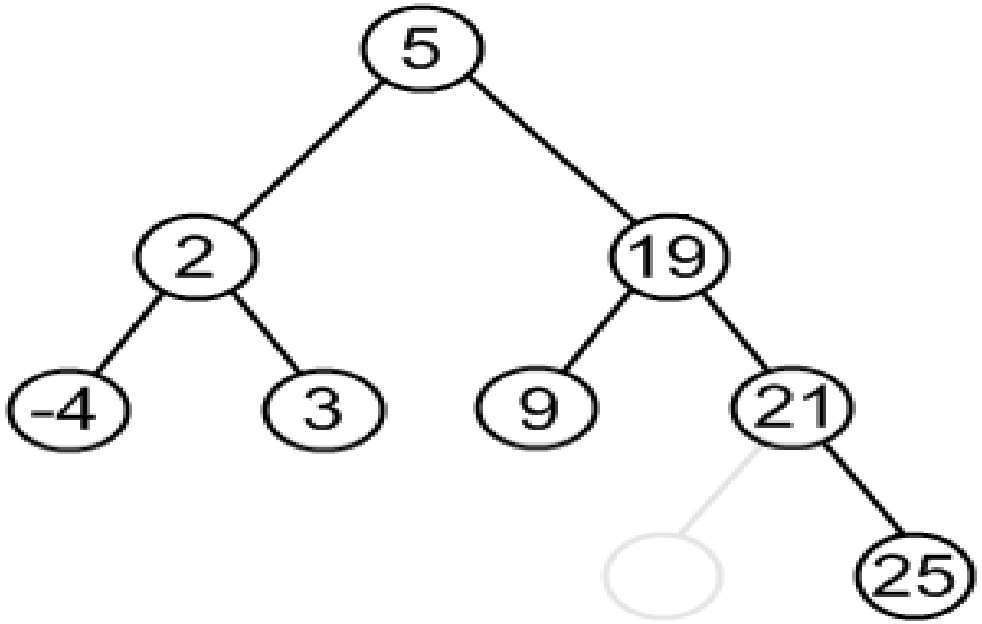
Lanj.

Copykan *Leaf* Paling Kiri Tersebut ke *Node* Yang Akan Dihapus



Lanj.

Musnahkan *Node* Yang Sudah Dicopykan Ke *Node* Yang Dihapus tersebut



Ringkasan

- Penghapusan *node* pada *binary tree* harus memenuhi kriteria sesuai dengan order penyusunannya
- Menghapus *node* yang merupakan *leaf* dapat dilakukan tanpa penanganan tambahan apapun
- Menghapus *node* yang memiliki satu anak hanya mmeerlukan perlakuan memindahkan *link ancestor* nya ke *node* dari anak *node* yang dihapus
- Menghapus *node* yang memiliki dua anak dapat dilakukan dengan dua perlakuan *Delete By Copying* dan *Delete By Merging*

Contoh procedure delete Node

```
void deleteNode(struct TheCell *travCell, int cari, struct TheCell *deleteCell)
{
  if (deleteCell==NULL)
  {
    //Cari node yang akan di hapus
    if (travCell->dat == cari)
    {
      cout<<"Data terdapat pada record ke "<<travCell->id<<endl;
      deleteNode(travCell, cari, travCell);
    }
```

Contoh procedure delete Node (lanj.)

```
  else
  {
    if (travCell->kiri != NULL)
      deleteNode(travCell->kiri, cari, NULL);
    if(travCell->kanan != NULL)
      deleteNode(travCell->kanan, cari, NULL);
  }
}
```

Contoh procedure delete Node (lanj.)

```
else
{
  if(deleteCell->kanan == NULL)
  {
    if (deleteCell->induk == NULL)
    {
      //berarti root
      rootCell = deleteCell->kiri;
    }
```

Contoh procedure delete Node (lanj.)

```
  else
  {
    //jika tidak punya anak disebelah kanan, tempatkan left sbg pengganti
    if(deleteCell->induk->kiri == deleteCell)
      deleteCell->induk->kiri = deleteCell->kiri;
    else if(deleteCell->induk->kanan == deleteCell)
      deleteCell->induk->kanan = deleteCell->kiri;
  }
}
```

Contoh procedure delete Node (lanj.)

```
else if (deleteCell->kiri == NULL)
{
  if (deleteCell->induk == NULL)
  {
    //berarti root
    rootCell = deleteCell->kanan;
  } else
  {
    //jika tidak punya anak disebela kiri, tempatkan right sbg pengganti
    if(deleteCell->induk->kiri == deleteCell)
      deleteCell->induk->kiri = deleteCell->kanan;
    else if(deleteCell->induk->kanan == deleteCell)
      deleteCell->induk->kanan = deleteCell->kanan;
  }
}
```

Contoh procedure delete Node (lanj.)

```
    else
    {
      if(deleteCell->induk->kiri == deleteCell)
        deleteCell->induk->kiri = deleteCell->kiri;
      else
        deleteCell->induk->kanan = deleteCell->kiri;
      travCell=deleteCell->kiri;
      while(travCell->kanan != NULL)
        travCell = travCell->kanan;
      travCell->kanan = deleteCell->kanan;
    }
    free(deleteCell);
  }
}
```

Terimakasih

TUHAN Memberkati Anda

Teady Matius Surya Mulyana (tmulyana@bundamulia.ac.id)