



Struktur Data

Pendidikan Teknik Informatika

Universitas Negeri Padang

TIM DOSEN ©2022

# JOBSHEET (JS-07)

## Multiway Search Tree

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Pendidikan Teknik Informatika	TIK1.61.2317	4 x 50 Menit

### TUJUAN PRAKTIKUM

1. Mahasiswa mampu mengaplikasikan konsep B tree dalam menyelesaikan kasus.
2. Mahasiswa mampu mengaplikasikan konsep B+ tree dalam menyelesaikan kasus.

### HARDWARE & SOFTWARE

1. Personal Computer
2. IDE: Dev C++

---

### TEORI SINGKAT

#### A. B Tree

##### 1. Definisi B Tree

B Tree adalah pohon m-way khusus yang dapat digunakan secara luas untuk akses disk. Sebuah B-Tree berorde m dapat memiliki paling banyak  $m-1$  kunci dan  $m$  anak. Salah satu alasan utama menggunakan B tree adalah kemampuannya untuk menyimpan sejumlah besar kunci dalam satu simpul dan nilai kunci yang besar dengan menjaga ketinggian pohon yang relatif kecil.

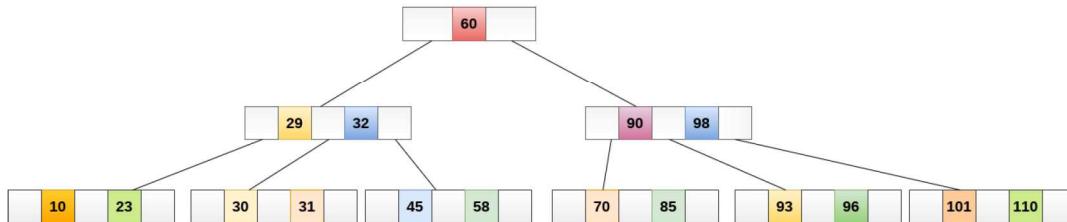
B tree orde m berisi semua properti pohon M-way. Selain itu, ini berisi properti berikut:

- a. Setiap node dalam B-Tree berisi paling banyak  $m$  anak.

- b. Setiap simpul dalam B-Tree kecuali simpul akar dan simpul daun mengandung setidaknya  $m/2$  anak.
- c. Node root harus memiliki minimal 2 node.
- d. Semua simpul daun harus berada pada level yang sama.

Tidak semua node harus memiliki jumlah anak yang sama, tetapi setiap node harus memiliki  $m/2$  jumlah node.

B tree orde 4 ditunjukkan pada gambar berikut :



Saat melakukan beberapa operasi pada B Tree, properti apa pun dari B Tree dapat melanggar seperti jumlah anak minimum yang dapat dimiliki sebuah node. Untuk mempertahankan sifat-sifat B tree, pohon dapat membelah atau bergabung.

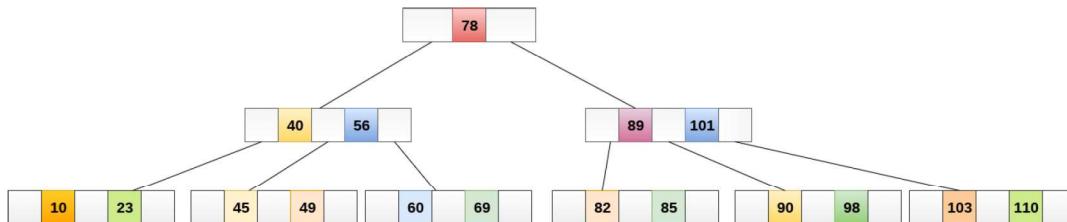
## 2. Operasi pada B Tree

### a. Pencarian (Searching)

Pencarian di B tree mirip dengan pencarian di pohon pencarian Biner. Pencarian di B tree tergantung pada ketinggian pohon.

Misalnya, jika kita mencari item 49 di B tree berikut, prosesnya akan seperti berikut:

1. Bandingkan item 49 dengan simpul akar 78. karena  $49 < 78$  maka pindah ke subpohon kirinya.
2. Karena,  $40 < 49 < 56$ , telusuri subpohon kanan dari 40.
3.  $49 > 45$ , pindah ke kanan. Bandingkan 49.
4. kecocokan ditemukan, kembali.



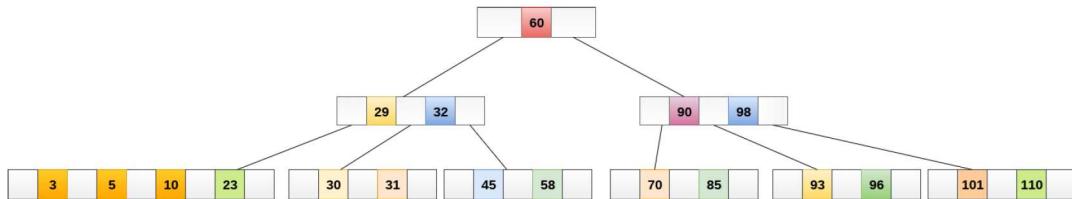
## b. Inserting

Penyisipan dilakukan pada tingkat simpul daun. Algoritma berikut perlu diikuti untuk memasukkan item ke dalam B Tree:

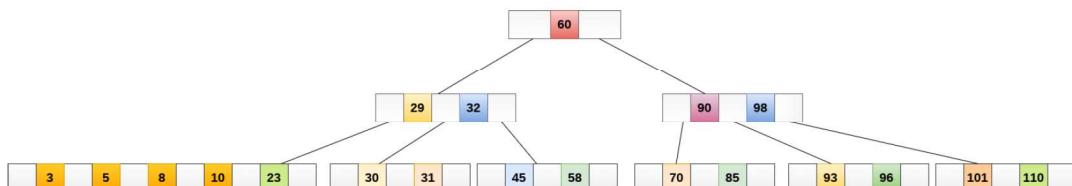
1. Lintasi B tree untuk menemukan simpul daun yang sesuai di mana simpul tersebut dapat disisipkan.
2. Jika simpul daun berisi kurang dari  $m-1$  kunci maka masukkan elemen dalam urutan yang meningkat.
3. Lain, jika simpul daun berisi  $m-1$  kunci, maka ikuti langkah-langkah berikut.
  - a) Masukkan elemen baru dalam urutan elemen yang meningkat.
  - b) Membagi node menjadi dua node di median.
  - c) Dorong elemen median ke atas ke simpul induknya.
  - d) Jika simpul induk juga berisi  $m-1$  jumlah kunci, maka pisahkan juga dengan mengikuti langkah yang sama.

Contoh:

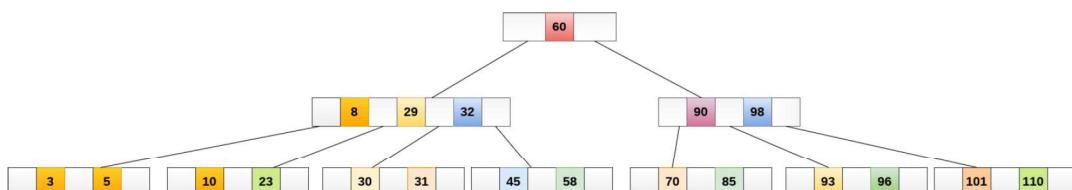
Masukkan simpul 8 ke dalam B tree orde 5 yang ditunjukkan pada gambar berikut :



8 akan disisipkan di sebelah kanan 5, oleh karena itu sisipkan 8.



Node, sekarang berisi 5 kunci yang lebih besar dari  $(5 - 1 = 4)$  kunci. Oleh karena itu pisahkan simpul dari median yaitu 8 dan dorong ke simpul induknya yang ditunjukkan sebagai berikut.



### c. Penghapusan (Deletion)

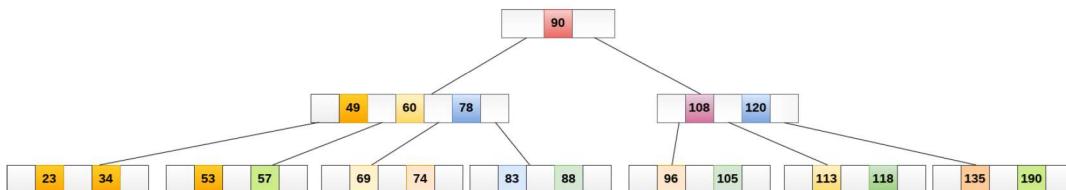
Penghapusan juga dilakukan pada simpul daun. Node yang akan dihapus dapat berupa leaf node atau internal node. Algoritma berikut perlu diikuti untuk menghapus simpul dari B tree:

1. Temukan simpul daun.
2. Jika ada lebih dari  $m/2$  kunci di simpul daun maka hapus kunci yang diinginkan dari simpul.
3. Jika simpul daun tidak berisi  $m/2$  kunci maka lengkapi kunci dengan mengambil elemen dari delapan atau saudara kiri.
  - a) Jika saudara kiri mengandung lebih dari  $m/2$  elemen, maka dorong elemen terbesarnya ke atas ke induknya dan pindahkan elemen perantara ke simpul di mana kuncinya dihapus.
  - b) Jika saudara kandung kanan mengandung lebih dari  $m/2$  elemen, maka dorong elemen terkecilnya ke atas ke induknya dan pindahkan elemen pengintervensi ke simpul di mana kuncinya dihapus.
4. Jika tidak satu pun dari saudara kandung mengandung lebih dari  $m/2$  elemen, buat simpul daun baru dengan menggabungkan dua simpul daun dan elemen perantara dari simpul induk.
5. Jika parent dibiarkan dengan kurang dari  $m/2$  node, terapkan proses di atas pada parent juga.

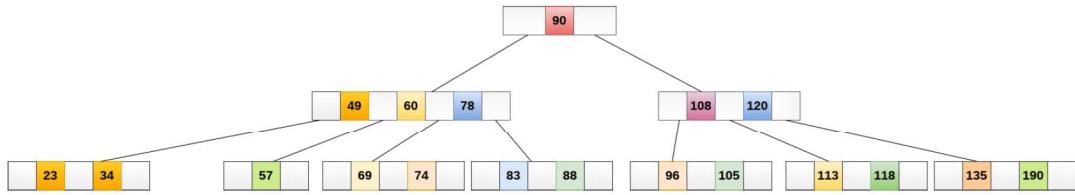
Jika node yang akan dihapus adalah node internal, maka ganti node tersebut dengan penerus atau pendahulunya secara berurutan. Karena, penerus atau pendahulu akan selalu berada di simpul daun, maka prosesnya akan sama seperti simpul yang dihapus dari simpul daun.

#### **Contoh :**

Hapus simpul 53 dari B tree orde 5 yang ditunjukkan pada gambar berikut:

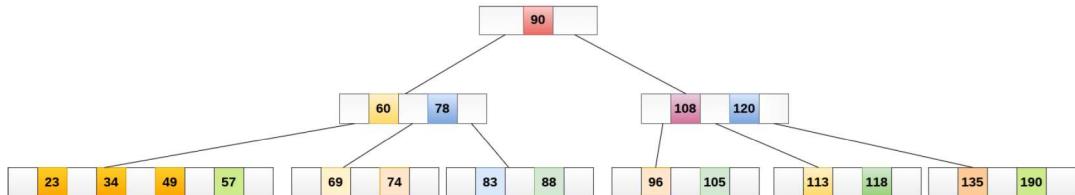


53 hadir di anak kanan elemen 49. Hapus itu.



Sekarang, 57 adalah satu-satunya elemen yang tersisa di simpul, jumlah minimum elemen yang harus ada di B tree orde 5, adalah 2. kurang dari itu, elemen di sub-pohon kiri dan kanannya juga tidak cukup oleh karena itu, gabungkan dengan saudara kandung kiri dan elemen intervensi induk yaitu 49.

B tree terakhir ditunjukkan sebagai berikut :



### 3. Penerapan B tree

B tree digunakan untuk mengindeks data dan menyediakan akses cepat ke data aktual yang disimpan di memori karena akses ke nilai yang disimpan dalam database besar yang disimpan di memori adalah proses yang sangat memakan waktu.

Mencari database yang tidak diindeks dan tidak disortir yang berisi n nilai kunci membutuhkan  $O(n)$  waktu berjalan dalam kasus terburuk. Namun, jika kita menggunakan B Tree untuk mengindeks database ini, database akan dicari dalam waktu  $O(\log n)$  dalam kasus terburuk.

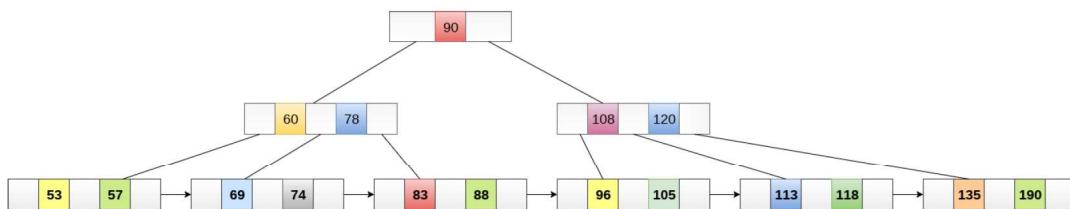
## B. B+ Tree

B+ Tree adalah perpanjangan dari B Tree yang memungkinkan operasi penyisipan, penghapusan, dan pencarian yang efisien. Di B Tree, Kunci dan catatan keduanya dapat disimpan di internal serta simpul daun. Sedangkan pada pohon B+, record (data) hanya dapat disimpan pada node daun sedangkan node internal hanya dapat menyimpan nilai

kunci. Node daun pohon B+ ditautkan bersama dalam bentuk daftar tertaut tunggal untuk membuat kueri penelusuran lebih efisien.

B+ Tree digunakan untuk menyimpan data dalam jumlah besar yang tidak dapat disimpan di memori utama. Karena fakta bahwa, ukuran memori utama selalu terbatas, node internal (kunci untuk mengakses catatan) dari pohon B+ disimpan di memori utama sedangkan node daun disimpan di memori sekunder.

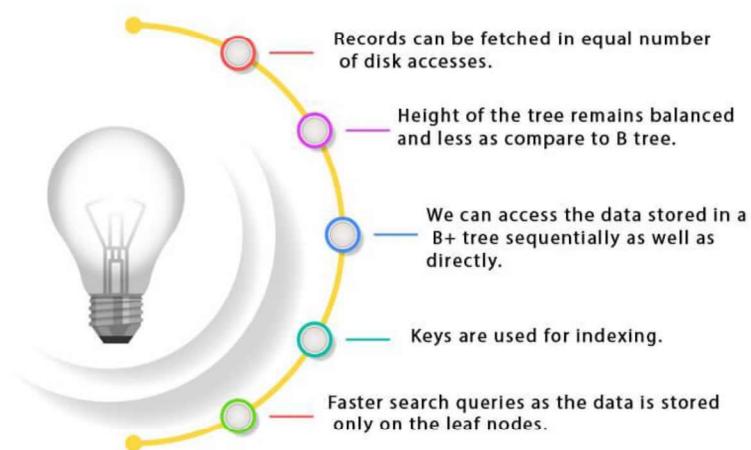
Node internal pohon B+ sering disebut node indeks. Sebuah pohon B+ orde 3 ditunjukkan pada gambar berikut :



### 1. Keuntungan dari B+ tree :

- Catatan dapat diambil dalam jumlah akses disk yang sama.
- Tinggi pohon tetap seimbang dan kurang dibandingkan dengan pohon B.
- Kita dapat mengakses data yang disimpan dalam pohon B+ secara berurutan maupun langsung.
- Kunci digunakan untuk pengindeksan.
- Queri pencarian lebih cepat karena data hanya disimpan di simpul daun.

### Advantages of B+ Tree



## 2. B Tree VS B+ Tree

SN	B Tree	B+ Tree
1	Kunci pencarian tidak dapat disimpan berulang kali.	Kunci pencarian yang berlebihan dapat ada.
2	Data dapat disimpan di node daun serta node internal	Data hanya dapat disimpan pada simpul daun.
3	Pencarian untuk beberapa data adalah proses yang lebih lambat karena data dapat ditemukan pada node internal serta pada node daun.	Pencarian relatif lebih cepat karena data hanya dapat ditemukan di simpul daun.
4	Penghapusan node internal sangat rumit dan memakan waktu.	Penghapusan tidak akan pernah menjadi proses yang kompleks karena elemen akan selalu dihapus dari simpul daun.
5	Node daun tidak dapat dihubungkan bersama.	Node daun dihubungkan bersama untuk membuat operasi pencarian lebih efisien.

## 3. Penyisipan (Insertion) di B+ Tree

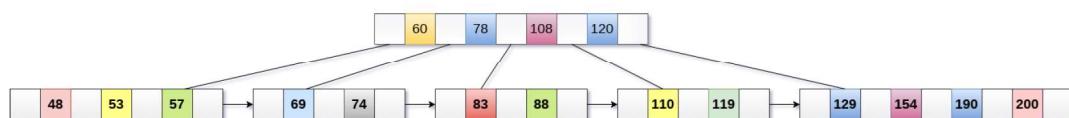
**Langkah 1:** Masukkan simpul baru sebagai simpul daun

**Langkah 2:** Jika daun tidak memiliki ruang yang diperlukan, pisahkan node dan salin node tengah ke node indeks berikutnya.

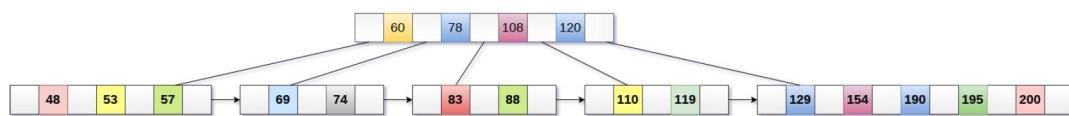
**Langkah 3:** Jika node indeks tidak memiliki ruang yang diperlukan, pisahkan node dan salin elemen tengah ke halaman indeks berikutnya.

**Contoh :**

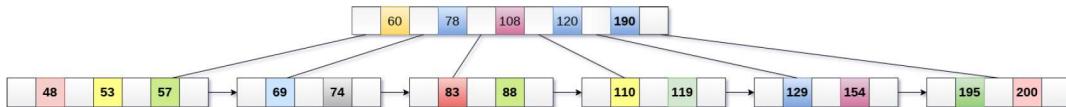
Masukkan nilai 195 ke dalam pohon B+ orde 5 yang ditunjukkan pada gambar berikut :



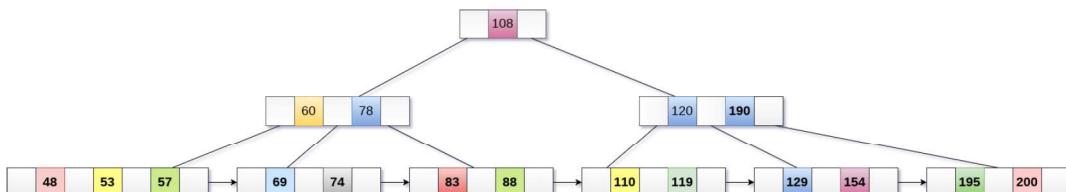
195 akan disisipkan di sub pohon kanan 120 setelah 190. Sisipkan pada posisi yang diinginkan.



Node berisi lebih besar dari jumlah maksimum elemen yaitu 4, oleh karena itu pisahkan dan tempatkan node median ke atas ke induknya.



Sekarang, node indeks berisi 6 anak dan 5 kunci yang melanggar properti pohon B+, oleh karena itu kita perlu membaginya, yang ditunjukkan sebagai berikut :



#### 4. Penghapusan di B+ Tree

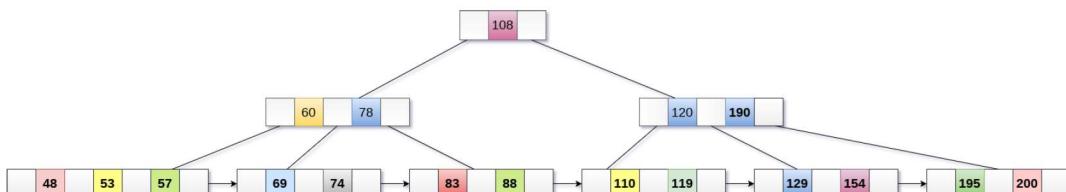
**Langkah 1:** Hapus kunci dan data dari daun.

**Langkah 2:** jika simpul daun berisi kurang dari jumlah minimum elemen, gabungkan simpul dengan saudaranya dan hapus kunci di antara mereka.

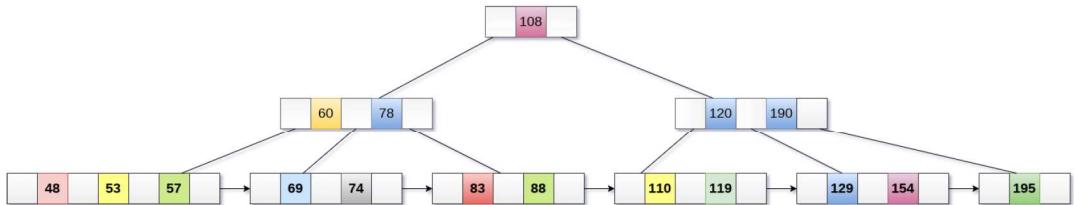
**Langkah 3:** jika node indeks berisi kurang dari jumlah elemen minimum, gabungkan node dengan saudaranya dan pindahkan kunci ke bawah di antara mereka.

**Contoh :**

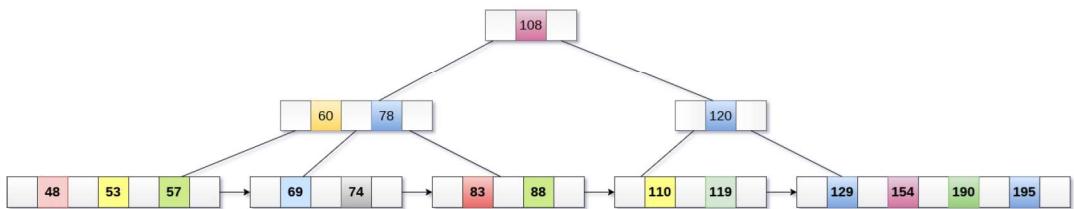
Hapus kunci 200 dari Pohon B+ yang ditunjukkan pada gambar berikut :



200 hadir di sub-pohon kanan 190, setelah 195. hapus.

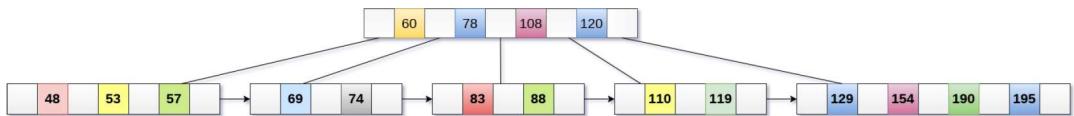


Gabungkan dua node dengan menggunakan 195, 190, 154 dan 129.



Sekarang, elemen 120 adalah elemen tunggal yang ada di node yang melanggar properti B+ Tree. Oleh karena itu, kita perlu menggabungkannya dengan menggunakan 60, 78, 108 dan 120.

Sekarang, tinggi pohon B+ akan berkurang 1.



## LATIHAN

### 1. B tree

```
/* Nama File : B tree
Pembuat : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

// Searching a key on a B-tree in C

#include <stdio.h>
#include <stdlib.h>

#define MAX 3
#define MIN 2

struct BTreenode {
```

```
int val[MAX + 1], count;
struct BTreenode *link[MAX + 1];
};

struct BTreenode *root;

// Create a node
struct BTreenode *createNode(int val, struct BTreenode
*child) {
    struct BTreenode *newNode;
    newNode = (struct BTreenode *)malloc(sizeof(struct
BTreenode));
    newNode->val[1] = val;
    newNode->count = 1;
    newNode->link[0] = root;
    newNode->link[1] = child;
    return newNode;
}

// Insert node
void insertNode(int val, int pos, struct BTreenode *node,
                struct BTreenode *child) {
    int j = node->count;
    while (j > pos) {
        node->val[j + 1] = node->val[j];
        node->link[j + 1] = node->link[j];
        j--;
    }
    node->val[j + 1] = val;
    node->link[j + 1] = child;
    node->count++;
}

// Split node
void splitNode(int val, int *pval, int pos, struct
BTreenode *node,
                struct BTreenode *child, struct BTreenode
**newNode) {
    int median, j;

    if (pos > MIN)
        median = MIN + 1;
    else
        median = MIN;
```

```

    *newNode = (struct BTreenode *)malloc(sizeof(struct
BTreenode));
    j = median + 1;
    while (j <= MAX) {
        (*newNode)->val[j - median] = node->val[j];
        (*newNode)->link[j - median] = node->link[j];
        j++;
    }
    node->count = median;
    (*newNode)->count = MAX - median;

    if (pos <= MIN) {
        insertNode(val, pos, node, child);
    } else {
        insertNode(val, pos - median, *newNode, child);
    }
    *pval = node->val[node->count];
    (*newNode)->link[0] = node->link[node->count];
    node->count--;
}

// Set the value
int setValue(int val, int *pval,
            struct BTreenode *node, struct BTreenode
**child) {
    int pos;
    if (!node) {
        *pval = val;
        *child = NULL;
        return 1;
    }

    if (val < node->val[1]) {
        pos = 0;
    } else {
        for (pos = node->count;
             (val < node->val[pos] && pos > 1); pos--)
        ;
        if (val == node->val[pos]) {
            printf("Duplicates are not permitted\n");
            return 0;
        }
    }
}

```

```
if (setValue(val, pval, node->link[pos], child)) {
    if (node->count < MAX) {
        insertNode(*pval, pos, node, *child);
    } else {
        splitNode(*pval, pval, pos, node, *child, child);
        return 1;
    }
}
return 0;
}

// Insert the value
void insert(int val) {
    int flag, i;
    struct BTreenode *child;

    flag = setValue(val, &i, root, &child);
    if (flag)
        root = createNode(i, child);
}

// Search node
void search(int val, int *pos, struct BTreenode *myNode) {
    if (!myNode)
        return;
}

if (val < myNode->val[1]) {
    *pos = 0;
} else {
    for (*pos = myNode->count;
        (val < myNode->val[*pos] && *pos > 1); (*pos)--)
    ;
    if (val == myNode->val[*pos]) {
        printf("%d is found", val);
        return;
    }
}
search(val, pos, myNode->link[*pos]);

return;
}

// Traverse then nodes
```

```

void traversal(struct BTreenode *myNode) {
    int i;
    if (myNode) {
        for (i = 0; i < myNode->count; i++) {
            traversal(myNode->link[i]);
            printf("%d ", myNode->val[i + 1]);
        }
        traversal(myNode->link[i]);
    }
}

int main() {
    int val, ch;

    insert(8);
    insert(9);
    insert(10);
    insert(11);
    insert(15);
    insert(16);
    insert(17);
    insert(18);
    insert(20);
    insert(23);

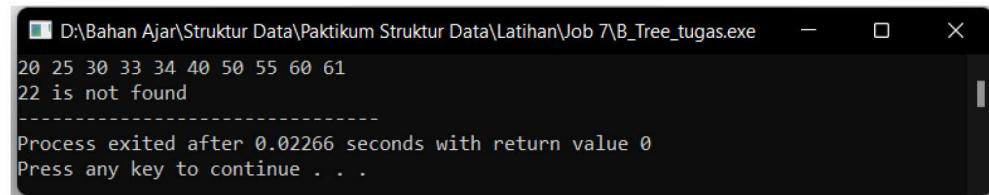
    traversal(root);

    printf("\n");
    search(11, &ch, root);
}

```

## TUGAS

Pahami dan modifikasilah program pada Latihan di atas sehingga menghasilkan sebuah program baru yang mempunyai output seperti tampilan di bawah ini :



```

D:\Bahan Ajar\Struktur Data\Paktikum Struktur Data\Latihan\Job 7\B_Tree_tugas.exe
20 25 30 33 34 40 50 55 60 61
22 is not found
-----
Process exited after 0.02266 seconds with return value 0
Press any key to continue . .

```

## DAFTAR PUSTAKA

- Kelley, Al and Pohl, Ira. 2003. *C by Dissection: The Essentials of C Programming*. Addison-Weasley.
- Khannedy, Eko Kurniawan. 2007. *Diktat Pemrograman C*. Bandung : Unikom.
- Liem, Inggriani. 2003. *Catatan Singkat Bahasa C*. Jurusan Teknik Informatika. Institut Teknologi Bandung.
- Reema Thareja. 2014. *Data Structures Using C*. India : Oxford University Press.
- Solichin, Achmad. 2003. *Pemrograman Bahasa C dengan Turbo C*. Kuliah Berseri ilmukomputer.com