KEGIATAN PRAKTIKUM 14

STRUKTUR DATA HIRARKI BAGIAN III (B TREE)

A. Perhatikan baris kode berikut:

1

```
/* Kode C++ untuk impelementasi B-Tree
          (dari berbagai sumber) */

#include <bits/stdc++.h>
using namespace std;

/* Nilai B (menyatakan ordo atau jumlah key yang diperbolehkan disimpan pada
1 node adalah N-1 */
#define N 4

struct node {
      int key[N - 1]; // Key dari N-1
      struct node* child[N]; // Array Child dari Panjang 'N'
      int isleaf; // Menyatakan posisi node pada leaf (1) atau tidak (0)
      int n; //Jumlah key dalam suatu node
      struct node* parent; //Menyatakan node parent
};
```

2

```
struct node* searchforleaf(struct node* root,
                           int k,struct node* parent,
                           int chindex){
      if (root) {
          if (... ... ... ...)// cek apakah root nerupakan leaf
               return root;

           else {// cek apakah memiliki 1 atau 2 anak
               int i;
               if (... < root->key[0])
                  root = searchforleaf(root->child[0], k, root, 0);
               else{
                  for (i = 0; i < root->n; i++)
                     if (root->key[i] > ...)
                        root = searchforleaf(root->child[i], k, root, i);
                  if (root->key[i - 1] < k)
                     root = searchforleaf(root->child[i], k, root, i);
                  }
           }
      }
      else {
            struct node* newleaf = new struct node;
            newleaf->isleaf = 1;
            newleaf->n = 0;
            parent->child[chindex] = newleaf;
            newleaf->parent = parent;
            return newleaf;
      }
}
```

```
struct node* insert(struct node* root, int k)
{
   if (root) {
      struct node* p = searchforleaf(root, k, NULL, 0);
      struct node* q = NULL;
      int e = k;

      for (int e = k; p; p = p->parent) {
          if (p->n == 0) {
              p->key[0] = e;
              p->n = 1;
              return root;}

          if (p->n < N - 1) {
              int i;
              for (i = 0; i < p->n; i++) {
                  if (p->key[i] > e) {
                     for (int j = p->n - 1; j >= i; j--)
                        p->key[j + 1] = p->key[j];
                     break;}
               }
               p->key[i] = e;
               p->n = p->n + 1;
               return root;
           }


           if (p->n == N - 1 && p->parent && p->parent->n < N) {
              int m;
              for (int i = 0; i < p->parent->n; i++)
                   if (p->parent->child[i] == p) {
                       m = i;
                       break;
                              }

              if (m + 1 <= N - 1){

                 q = p->parent->child[m + 1];
                 if (q) {

                     if (q->n == N - 1) {
                         struct node* r = new struct node;
                         int* z = new int[((2 * N) / 3)];
                         int parent1, parent2;
                         int* marray = new int[2 * N];
                         int i;
                         for (i = 0; i < p->n; i++)
                            marray[i] = p->key[i];
                         int fege = i;
                         marray[i] = e;
                         marray[i + 1] = p->parent->key[m];
                         for (int j = i + 2; j < ((i + 2) + (q->n)); j++)
                            marray[j] = q->key[j - (i + 2)];
```

```
                        for (int i = 0; i < (2 * N - 2) / 3; i++)
                            p->key[i] = marray[i];
                        parent1 = marray[(2 * N - 2) / 3];

                        for (int j =((2*N-2) / 3) + 1; j <(4 * N) / 3; j++)
                            q->key[j - ((2 * N - 2) / 3 + 1)] = marray[j];
                        parent2 = marray[(4 * N) / 3];

                        for (int f = ((4 * N) / 3 + 1); f < 2 * N; f++)
                            r->key[f - ((4 * N) / 3 + 1)] = marray[f];

                        if (m == 0 || m == 1) {
                            p->parent->key[0] = parent1;
                            p->parent->key[1] = parent2;
                            p->parent->child[0] = p;
                            p->parent->child[1] = q;
                            p->parent->child[2] = r;
                            return root;}
                        else {
                            p->parent->key[m - 1] = parent1;
                            p->parent->key[m] = parent2;
                            p->parent->child[m - 1] = p;
                            p->parent->child[m] = q;
                            p->parent->child[m + 1] = r;
                            return root;
                              }
                        }
                    }
                    else
                    {
                       int put;
                       if (m == 0 || m == 1)
                          put = p->parent->key[0];
                       else
                          put = p->parent->key[m - 1];
                       for (int j = (q->n) - 1; j >= 1; j--)
                          q->key[j + 1] = q->key[j];
                       q->key[0] = put;
                       p->parent->key[m == 0 ? m : m - 1] = p->key[p->n-1];
                    }
                 }
              }
           }
        }
        else
        {
           // Buat node baru jika Root NULL
           struct node* root = new struct node;
           root->key[0] = k;
           root->isleaf = 1;
           root->n = 1;
           root->parent = NULL;
        }
}
```

```
// Lihat hasil B-Tree
int main(){
    /* Jika Tree beriut ini sudah dikonstruksi
    Dengan Root 6 dan memiliki anak kiri (1, 2, 4) dam kanan (7,8,9)
                  6
              /       \
           1 2 4     7 8 9

    Kemudian ditambahkan elemen nilai 5, dan B-Tree menjadi:
                  4     7
              /   \    \
            1 2   5 6   8 9
    */
    // Mulai dengan Empty Root
    struct node* ... ... ... ...;
    // Tambahkan 6
    root = ... ... ... ... (root, 6);

    // Tambahkan  1, 2, 4 Ke kiri dari 6
    root->child[0] = ... ...(... ... ... ..., 2);
    ... ... ... ... = insert(root->child[0], ...);
    root->child[0] = insert(... ... ... ...);
    root->child[0]->parent = root;


    // Tambahkan 7, 8, 9 Ke kanan dari 6
    root->child[1] = insert(... ... ... ...);
    ... ... ... ... = insert(root->child[1], ...);
    root->child[1] = ... ... ... ... , 7);
    root->... ... ... ... = root;

    // Cetak Tree Original
    cout << "Original tree: " << endl;
    for (int ... ..; ... < root->n; ... ...)
        cout << "      "<< root->key[z] << " ";
    cout << endl;
    for (int ... ... ; ... < N-1; ... ...) {
        cout << root->child[t]->key[0] << " ";
        cout << root->child[t]->key[1] << " ";
        cout << root->child[t]->key[2] << " ";
        cout << "   "
    }
    cout << endl;

    // Tambahkan Nilai 5
    cout << "Setelah menambakan ima 5: " << endl;
    root->child[0] = insert(root->child[0], 5);

    // Cetak Tree hasil penambahan dengan elemen 5
    ... ... ... ... ... ... ... ... ... ... ... ...
    ... ... ... ... ... ... ... ... ... ... ... ...
    ... ... ... ... ... ... ... ... ... ... ... ...
    return 0;
}
```

Berdasarkan kode program di atas, jawab pertanyaan berikut ini..

1. Lengkapi potongan kode (snippet) nomor 2, jelaskan hasil eksekusi dari baris kode tersebut?

2. Berikan Gambaran hasil eksekusi dari baris kode (snippet) nomor 3? Berikan ***highlight/block*** dari potongan baris dan kaitkan dengan penjelasan dari salah satu karateristik algoritme B-Tree

3. Lengkapi potongan kode (snippet) nomor 4, dan implementasikan keseluruhan kode agar dapat menjalankan perintah kode B-Tree dan perlihatkan hasilnya

4. Tuliskan *driver code* (***int main***) dengan memodifikasi baris kode pada nomor 4 yang memperlihatkan bentuk B-Tree yang sudah dikonstruksi:
   Dengan Node root adalah 9, dan 24. Kemudian memiliki anak kiri bagi 9 (7, 6, 8), anak kanan bagi 9 atau anak kiri bagi 24 (12, 17 19), dan anak kanan bagi 24 (29, 31, 41)
   Cetak hasilnya dan perlihatkan hasil ***printscreen***

5. Tuliskan *driver code* (***int main***) dengan memodifikasi baris kode pada nomor 4 yang memperlihatkan bentuk B-Tree yang sudah dikonstruksi:

   Dengan Node root adalah 14 dan memiliki anak kiri (12, 2, 9) dam kanan (49, 21, 31)

   ```
          14
        /    \
   2 9 12     21 31 49
   ```

   Jika ditambahkan elemen nilai 17, Cetak hasilnya dan perlihatkan hasil ***printscreen*** dari penambahan elemen tersebut.