MODUL 10
GRAPH DAN TREE

A. TUJUAN PRAKTIKUM

a. Mahasiswa diharapkan mampu memahami graph dan tree
b. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

B. DASAR TEORI

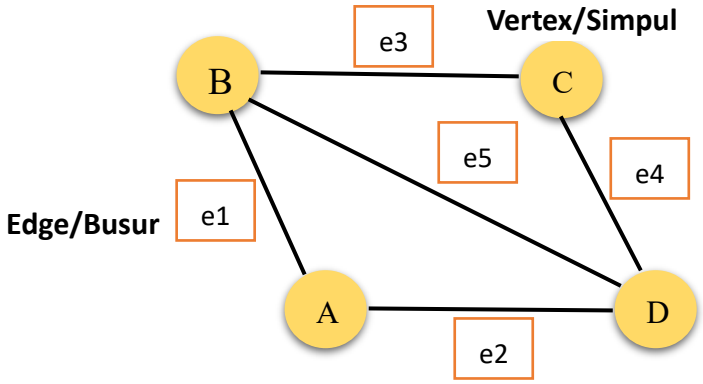
1. Graph

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde.

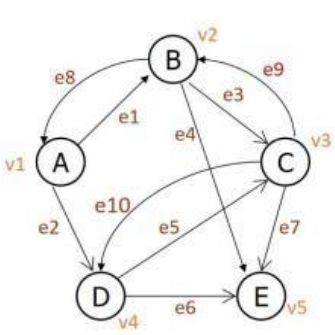
Dapat digambarkan:



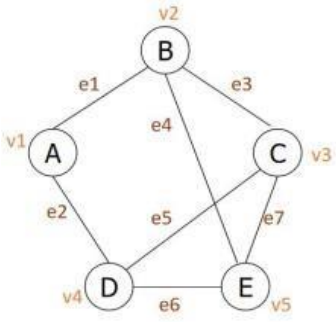
Gambar 1 Contoh Graph

Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

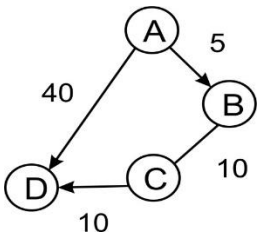
Jenis-jenis Graph



Directed Graph



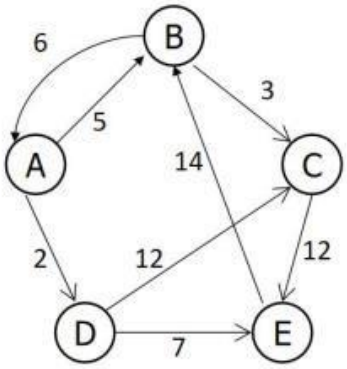
Undirected Graph



Weight Graph

a. **Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
b. **Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
c. **Weigth Graph :** Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks

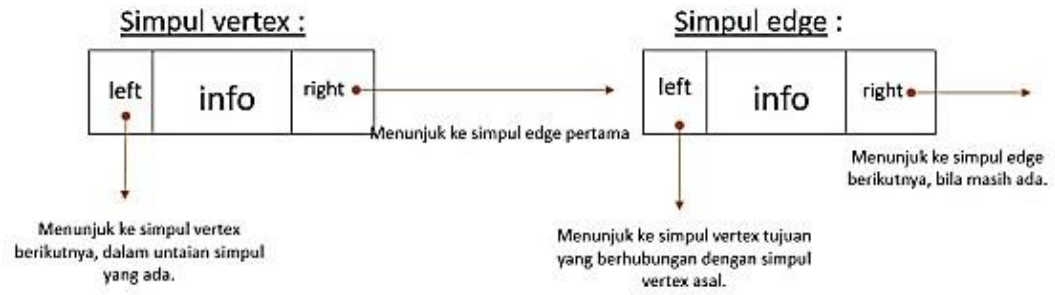


Ke

Dari		A 0	B 1	C 2	D 3	E 4
A	0	0	5	0	2	0
B	1	6	0	3	0	0
C	2	0	0	0	0	9
D	3	0	0	12	0	7
E	4	0	14	0	0	0

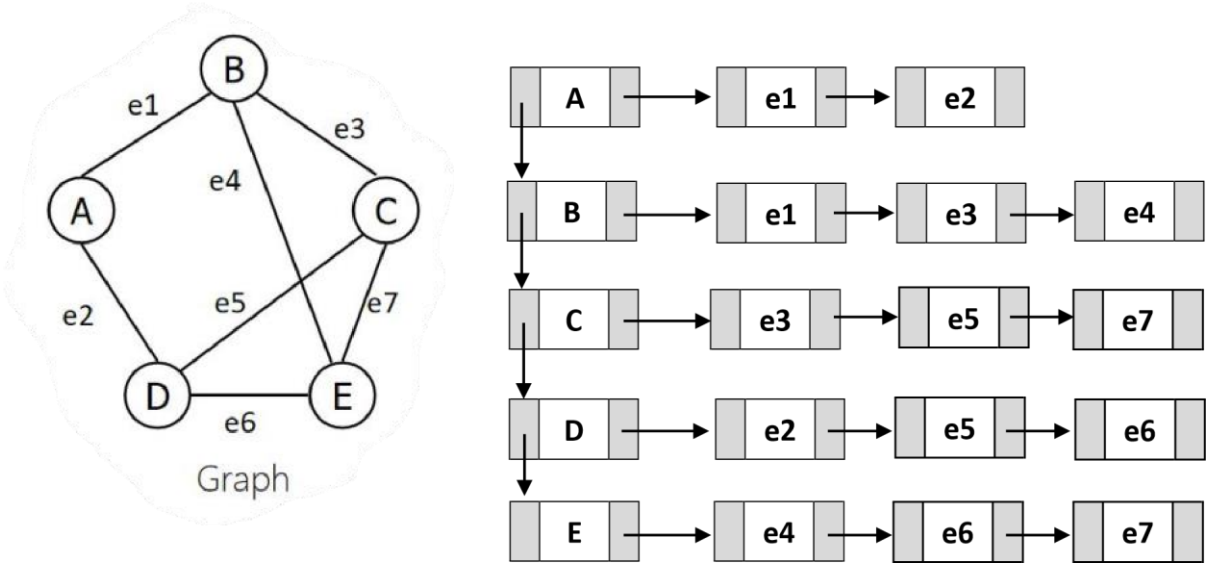
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

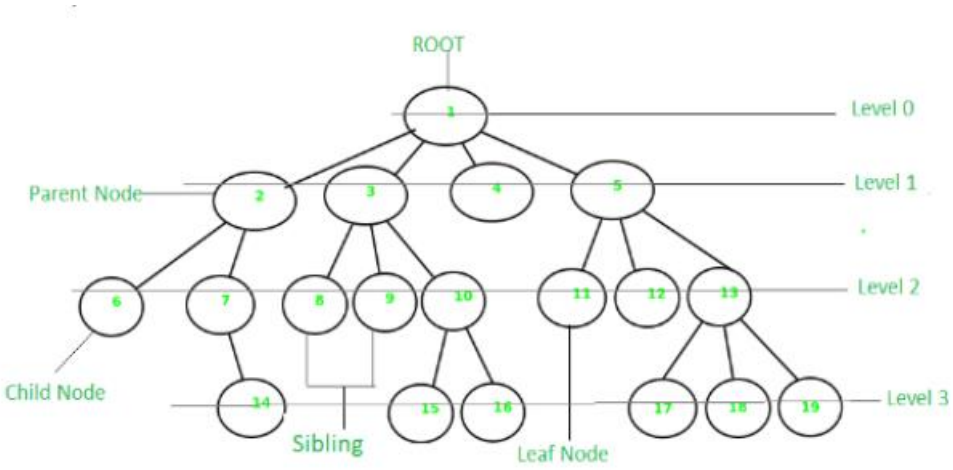
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, namun biasanya disamakan. Yang membedakan antara simpul vertex dengan simpul edge nantinya adalah anggapan terhadap simpul tersebut juga fungsinya masing-masing.



Gambar 6 Representasi Graph dengan Linked List

2. Tree atau Pohon

Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :

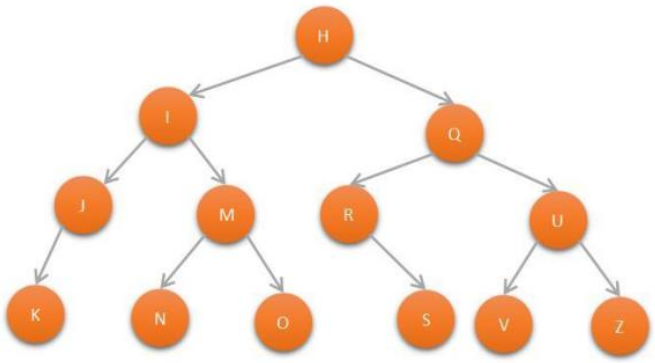


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Roof	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Tabel 1 Terminologi dalam Struktur Data Tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child)

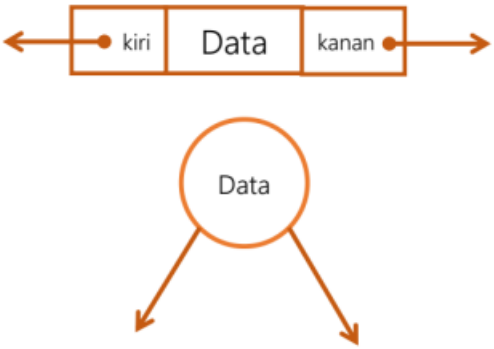
tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



Gambar 1 Struktur Data Binary Tree

Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```
struct pohon{
    char data;
    pohon *kanan;
    pohon *kiri;
};
pohon *simpul;
```



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

a. **Create**: digunakan untuk membentuk binary tree baru yang masih kosong.
b. **Clear**: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
c. **isEmpty**: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
d. **Insert**: digunakan untuk memasukkan sebuah node kedalam tree.
e. **Find**: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong**.**
f. **Update**: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

g. **Retrive**: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

h. **Delete Sub**: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

i. **Characteristic**: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.

j. **Traverse**: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

1. **Pre-Order**

   Penelusuran secara pre-order memiliki alur:

   a. Cetak data pada simpul root

   b. Secara rekursif mencetak seluruh data pada subpohon kiri

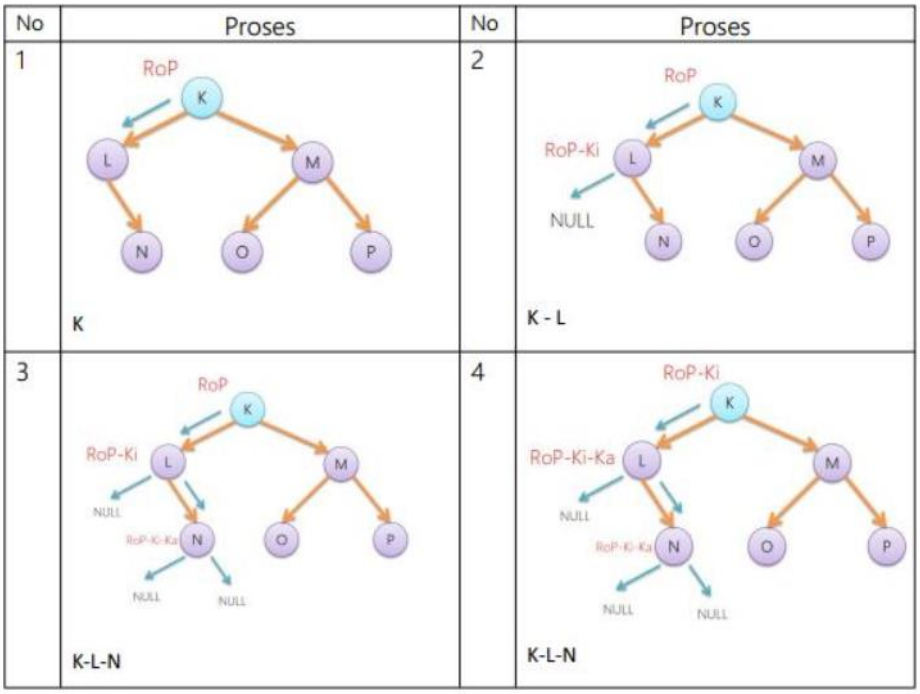
   c. Secara rekursif mencetak seluruh data pada subpohon kanan

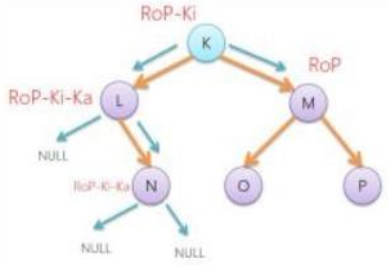
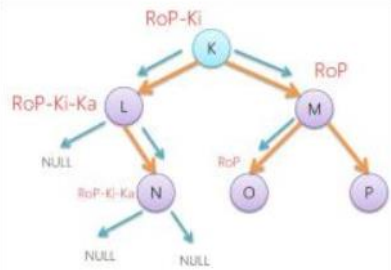
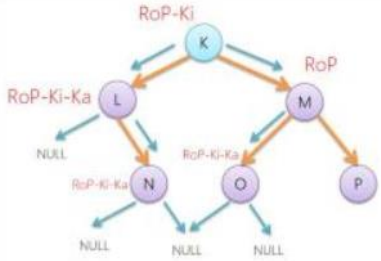
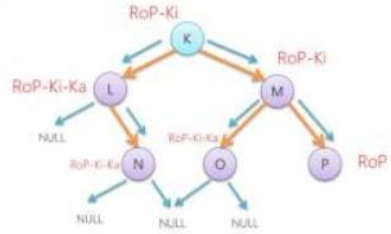
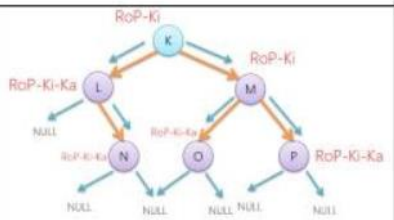
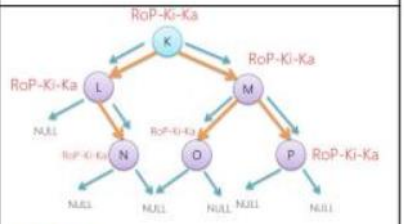
   Dapat kita turunkan rumus penelusuran menjadi:

   Root (print) - Kiri - Kanan

   RoP - Ki - Ka

   Alur pre-order



5	 K-L-N-M	6	 K-L-N-M-O
7	 K-L-N-M-O	8	 K-L-N-M-O-P
9	 K-L-N-M-O-P	10	 K-L-N-M-O-P

2. In-Order

Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri

b. Cetak data pada root

c. Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

Ro - KiP - Ka

3. Post Order

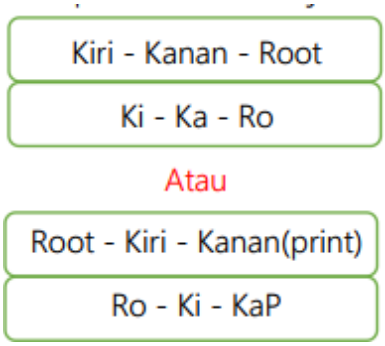
Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri

b. Secara rekursif mencetak seluruh data pada subpohon kanan

c. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



C. UNGUIDED

Program Graph

```
#include <iostream>
#include <iomanip>
using namespace std;

string simpul[7] ={"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogjakarta"};
int busur[7][7] =
{
    {0,7,8,0,0,0,0},
    {0,0,5,0,0,15,0},
    {0,6,0,0,5,0,0},
    {0,5,0,0,2,4,0},
    {23,0,0,10,0,0,8},
    {0,0,0,0,7,0,3},
    {0,0,0,0,9,4,0}};

void tampilGraph()
{
    for(int baris =0; baris <7; baris++){
        cout << " "  <<setiosflags(ios::left)<<setw(15)
        <<simpul[baris]  << " : ";
        for(int kolom=0;kolom<7;kolom++){
            if(busur[baris][kolom] !=0){
                cout << " " << simpul[kolom] << "("
```

```
                << busur[baris][kolom] << ")";
            }
        }cout << endl;
    }}

int main()
{
    tampilGraph();
    return 0;
}
```

Program Tree

```
#include <iostream>

using namespace std;

///PROGRAM BINARY TREE

//Deklarasi Pohon
struct Pohon{
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root, *baru;

//Inisialisasi
void init()
{
    root = NULL;
}

//Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1;   //true
    else
        return 0;   //false
}

//Buat Node Baru
void buatNode(char data )
{
  if(isEmpty() == 1){
    root = new Pohon();
    root->data = data;
    root->left = NULL;
    root->right = NULL;
    root->parent = NULL;
    cout << "\n Node " << data << " berhasil dibuat menjadi root."
<< endl;
  }
  else{
    cout << "\n Pohon sudah dibuat" << endl;
  }
}

//Tambah Kiri
```

```
Pohon *insertLeft(char data, Pohon *node )
{
  if(isEmpty() == 1){
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
  }else{
    // cek apakah child kiri ada atau tidak
    if( node->left != NULL ){
      // kalau ada
        cout << "\n Node "<< node->data << " sudah ada child kiri!"
<< endl;
        return NULL;
    }else{
        // kalau tidak ada
        baru = new Pohon();
        baru->data = data;
        baru->left = NULL;
        baru->right = NULL;
        baru->parent = node;
        node->left = baru;
        cout << "\n Node " << data << " berhasil ditambahkan ke
child kiri " << baru->parent->data << endl;
        return baru;
    }
  }
}

//Tambah Kanan
Pohon *insertRight(char data, Pohon *node )
{
  if( root == NULL ){
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
  }else{
    // cek apakah child kanan ada atau tidak
    if( node->right != NULL ){
      // kalau ada
      cout << "\n Node " << node->data << " sudah ada child kanan!"
<< endl;
      return NULL;
    }else{
      // kalau tidak ada
      baru = new Pohon();
      baru->data = data;
      baru->left = NULL;
      baru->right = NULL;
      baru->parent = node;
      node->right = baru;
      cout << "\n Node " << data << " berhasil ditambahkan ke child
kanan " << baru->parent->data << endl;
      return baru;
    }
  }
}

// Ubah Data Tree
void update(char data, Pohon *node)
{
  if(isEmpty() == 1){
    cout << "\n Buat tree terlebih dahulu!" << endl;
  }else{
```

```
    if( !node )
      cout << "\n Node yang ingin diganti tidak ada!!" << endl;
    else{
      char temp = node->data;
      node->data = data;
      cout << "\n Node " << temp << " berhasil diubah menjadi " <<
data << endl;
    }
  }
}

// Lihat Isi Data Tree
void retrieve( Pohon *node )
{
  if( !root ){
    cout << "\n Buat tree terlebih dahulu!" << endl;
  }else{
    if( !node )
      cout << "\n Node yang ditunjuk tidak ada!" << endl;
    else{
      cout << "\n Data node : " << node->data << endl;
    }
  }
}

// Cari Data Tree
void find(Pohon *node)
{
  if( !root ){
    cout << "\n Buat tree terlebih dahulu!" << endl;
  }else{
    if( !node )
      cout << "\n Node yang ditunjuk tidak ada!" << endl;
    else{
      cout << "\n Data Node : " << node->data << endl;
      cout << " Root : " << root->data << endl;

      if( !node->parent )
        cout << " Parent : (tidak punya parent)" << endl;
      else
        cout << " Parent : " << node->parent->data << endl;


      if( node->parent != NULL && node->parent->left != node &&
node->parent->right == node )

        cout << " Sibling : " << node->parent->left->data << endl;

      else if( node->parent != NULL && node->parent->right != node
&&
      node->parent->left == node )

        cout << " Sibling : " << node->parent->right->data << endl;

      else

        cout << " Sibling : (tidak punya sibling)" << endl;


      if( !node->left )
        cout << " Child Kiri : (tidak punya Child kiri)" << endl;
```

```
      else
        cout << " Child Kiri : " << node->left->data << endl;

      if( !node->right )
        cout << " Child Kanan : (tidak punya Child kanan)" << endl;
      else
        cout << " Child Kanan : " << node->right->data << endl;
    }
  }
}

// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
  if(!root)
    cout << "\n Buat tree terlebih dahulu!" << endl;
  else{

    if( node != NULL ){
      cout << " " << node->data << ", ";
      preOrder(node->left);
      preOrder(node->right);
    }
  }
}

// inOrder
void inOrder(Pohon *node = root)
{
  if(!root)
    cout << "\n Buat tree terlebih dahulu!" << endl;
  else{

    if(node != NULL){
      inOrder(node->left);
      cout << " " << node->data << ", ";
      inOrder(node->right);
    }
  }
}

// postOrder
void postOrder(Pohon *node = root)
{
  if(!root)
    cout << "\n Buat tree terlebih dahulu!" << endl;
  else{
    if( node != NULL ){
      postOrder(node->left);
      postOrder(node->right);
      cout << " " << node->data << ", ";
    }
  }
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{
  if(!root)
    cout << "\n Buat tree terlebih dahulu!" << endl;
```

```
  else{
    if( node != NULL ){
      if( node != root ){
        node->parent->left = NULL;
        node->parent->right = NULL;
      }
      deleteTree(node->left);
      deleteTree(node->right);

      if( node == root ){
        delete root;
        root = NULL;
      }else{
        delete node;
      }
    }
  }
}

// Hapus SubTree
void deleteSub(Pohon *node){
  if( !root )
    cout << "\n Buat tree terlebih dahulu!" << endl;
  else{
    deleteTree(node->left);
    deleteTree(node->right);
    cout << "\n Node subtree " << node->data << " berhasil
dihapus."  << endl;
  }
}

// Hapus Tree
void clear(){
  if( !root )
    cout << "\n Buat tree terlebih dahulu!!" << endl;
  else{
    deleteTree(root);
    cout << "\n Pohon berhasil dihapus."  << endl;
  }
}

// Cek Size Tree
int size(Pohon *node = root){
  if( !root ){
    cout << "\n Buat tree terlebih dahulu!!" << endl;
    return 0;
  }else{

    if( !node ){
      return 0;
    }else{
      return 1 + size( node->left ) + size(node->right);
    }

  }
}

// Cek Height Level Tree
int height( Pohon *node = root )
{
  if( !root ){
```

```
      cout << "\n Buat tree terlebih dahulu!" << endl;
      return 0;
    }else{
      if( !node ){
        return 0;
      }else{
        int heightKiri = height( node->left );
        int heightKanan = height( node->right );

        if( heightKiri >= heightKanan ){
          return heightKiri + 1;
        }else{
          return heightKanan + 1;
        }

      }
    }
}

// Karakteristik Tree
void charateristic()
{
  cout << "\n Size Tree : " << size() << endl;
  cout << " Height Tree : " << height() << endl;
  cout << " Average Node of Tree : " << size() / height() << endl;
}

int main()
{
  buatNode('A');

  Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
*nodeI, *nodeJ;

  nodeB = insertLeft('B', root);
  nodeC = insertRight('C', root);
  nodeD = insertLeft('D', nodeB);
  nodeE = insertRight('E', nodeB);
  nodeF = insertLeft('F', nodeC);
  nodeG = insertLeft('G', nodeE);
  nodeH = insertRight('H', nodeE);
  nodeI = insertLeft('I', nodeG);
  nodeJ = insertRight('J', nodeG);

  update('Z', nodeC);
  update('C', nodeC);

  retrieve(nodeC);

  find(nodeC);

  cout << "\n PreOrder :" << endl;
  preOrder(root);
  cout << "\n" << endl;
  cout << " InOrder :" << endl;
  inOrder(root);
  cout << "\n" << endl;
  cout << " PostOrder :" << endl;
  postOrder(root);
  cout << "\n" << endl;
```

```
    charateristic();

    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n" << endl;

    charateristic();

}
```

D. GUIDED

*Cantumkan NIM pada salah satu variabel di dalam program.

Contoh : int nama_22102003;

1. Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

   Output Program

```
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukkan bobot antar simpul
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0

         BALI     PALU
  BALI      0     3
  PALU      4     0

Process returned 0 (0x0)   execution time : 11.763 s
Press any key to continue.
```

2. Modifikasi unguided tree diatas dengan program menu menggunakan input data tree dari user!