

**LAPORAN PRAKTIKUM  
ALGORITMA & STRUKTUR DATA  
MODUL 7**



**Tree (Pohon)**

**Oleh:**

**Rizki Adhitiya Maulana**

**NIM. 2410817110014**

**PROGRAM STUDI TEKNOLOGI INFORMASI  
FAKULTAS TEKNIK  
UNIVERSITAS LAMBUNG MANGKURAT  
JUNI 2025**

**LEMBAR PENGESAHAN**  
**LAPORAN PRAKTIKUM ALGORITMA & STRUKTUR DATA**  
**MODUL 7**

Laporan Praktikum Algoritma & Struktur Data Modul 7 : Tree (Pohon) ini disusun sebagai syarat lulus mata kuliah Praktikum Algoritma & Struktur Data. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Rizki Adhitiya Maulana  
NIM : 2410817110014

Menyetujui,  
Asisten Praktikum

Mengetahui,  
Dosen Penanggung Jawab Praktikum

Muhammad Fauzan Ahsani  
NIM. 2310817310009

Muti'a Maulida, S.Kom., M.TI.  
NIP. 198810272019032013

## DAFTAR ISI

LEMBAR PENGESAHAN .....	i
DAFTAR ISI.....	ii
DAFTAR TABEL .....	iii
DAFTAR GAMBAR .....	iv
SOAL 1 .....	1
A    Source Code .....	4
B    Output Program.....	10
C    Pembahasan.....	12
TAUTAN GITHUB .....	18

## DAFTAR TABEL

Tabel 1 Source Code .....	4
---------------------------	---

## DAFTAR GAMBAR

Gambar 1 Soal 1 Modul 7 .....	3
Gambar 2 Tampilan Menu Tree .....	10
Gambar 3 Tampilan Menu Insert Tree .....	10
Gambar 4 Tampilan Menu PreOrder Tree .....	10
Gambar 5 Tampilan Menu InOrder Tree .....	11
Gambar 6 Tampilan Menu PostOrder Tree .....	11
Gambar 7 Tampilan Menu Exit .....	11
Gambar 8 Ilustrasi Insertion Pada Tree .....	14
Gambar 9 Ilustrasi PreOrder Pada Tree .....	15
Gambar 10 Ilustrasi InOrde Pada Tree .....	16
Gambar 11 ilustrasi PostOrder Pada Tree .....	17

## SOAL 1

Cobalah program berikut, perbaiki output, lengkapi fungsi inOrder dan postOrder pada coding, running, simpan program !

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <stdlib.h>
4  #include <iostream>
5
6  using namespace std;
7  struct Node
8  {
9      int data;
10     Node *kiri;
11     Node *kanan;
12 };
13
14 void tambah(Node **root, int databaru)
15 {
16     if (*root == NULL)
17     {
18         Node *baru;
19         baru = new Node;
20         baru->data = databaru;
21         baru->kiri = NULL;
22         baru->kanan = NULL;
23         (*root) = baru;
24         (*root)->kiri = NULL;
25         (*root)->kanan = NULL;
26         cout << "Data bertambah";
27     }
28     else if (databaru < (*root)->data)
29         tambah(&(*root)->kiri, databaru);
30     else if (databaru > (*root)->data)
31         tambah(&(*root)->kanan, databaru);
32     else if (databaru == (*root)->data)
33         cout << "Data sudah ada";
34 }
35
36 void preOrder(Node *root)
37 {
38     if (root != NULL)
39     {
40         cout << root->data;
41         preOrder(root->kiri);
42         preOrder(root->kanan);
43     }
44 }
45
46 void inOrder(Node *root)
47 {
48     if (root != NULL)
```

```

49
50
51
52
53
54 }
55
56 void postOrder(Node *root)
57 {
58
59
60
61
62
63
64 }
65
66 int main()
67 {
68     int pil, data;
69     Node *pohon;
70     pohon = NULL;
71     do
72     {
73         system("cls");
74         cout << "1. Tambah\n";
75         cout << "2. PreOrder\n";
76         cout << "3. inOrder\n";
77         cout << "4. PostOrder\n";
78         cout << "5. Exit\n";
79         cout << "\nPilihan : ";
80         cin >> pil;
81         switch (pil)
82         {
83             case 1:
84                 cout << "\n INPUT : ";
85                 cout << "\n -----";
86                 cout << "\n Data baru : ";
87                 cin >> data;
88                 tambah(&pohon, data);
89                 break;
90             case 2:
91                 cout << "PreOrder";
92                 cout << "\n-----\n";
93                 if (pohon != NULL)
94                 {
95                     preOrder(pohon);
96                 }

```

```

97         else
98             cout << "Masih Kosong";
99         break;
100     case 3:
101         cout << "InOrder";
102         cout << "\n-----\n";
103         if (pohon != NULL)
104         {
105             inOrder(pohon);
106         }
107         else
108             cout << "Masih Kosong";
109         break;
110     case 4:
111         cout << "PostOrder";
112         cout << "\n-----\n";
113         if (pohon != NULL)
114         {
115             postOrder(pohon);
116         }
117         else
118             cout << "Masih Kosong";
119         break;
120     case 5:
121         return 0;
122     }
123     _getch();
124 } while (pil != 5);
125 return EXIT_FAILURE;
126 }

```

Gambar 1 Soal 1 Modul 7



**A Source Code***Tabel 1 Source Code*

1	#include <iostream>
2	#include <conio.h>
3	#include <stdlib.h>
4	
5	using namespace std;
6	
7	struct Node
8	{
9	int data;
10	Node *left;
11	Node *right;
12	};
13	
14	void insert(Node **root, int newData)
15	{
16	if (*root == nullptr)
17	{
18	Node *newNode;
19	newNode = new Node;
20	
21	newNode -> data = newData;
22	newNode -> left = nullptr;
23	newNode -> right = nullptr;
24	
25	*root = newNode;
26	
27	cout << " Data has been added";
28	}

```
29
30     else if (newData < (*root) -> data)
31     {
32         insert(&((*root)->left), newData);
33     }
34
35     else if (newData > (*root) -> data)
36     {
37         insert(&((*root)->right), newData);
38     }
39
40     else if (newData == (*root) -> data)
41     {
42         cout << " Data is already exist";
43     }
44 }
45
46 void preOrder(Node *root)
47 {
48     if (root == nullptr) return;
49     cout << root->data << " ";
50     preOrder(root->left);
51     preOrder(root->right);
52 }
53
54 void inOrder(Node *root)
55 {
56     if (root == nullptr) return;
57     inOrder(root->left);
58     cout << root->data << " ";
```

```
59     inOrder(root->right);
60 }
61
62 void postOrder(Node *root)
63 {
64     if (root == nullptr) return;
65     postOrder(root->left);
66     postOrder(root->right);
67     cout << root->data << " ";
68 }
69
70 // side quest
71 void printTree() {
72 }
73
74 void freeTree(Node *root)
75 {
76     if (root == nullptr) return;
77     freeTree(root->left);
78     freeTree(root->right);
79     delete root;
80 }
81
82 int main()
83 {
84     int opt, val;
85     Node *tree;
86     tree = nullptr;
87
88     do
```

```

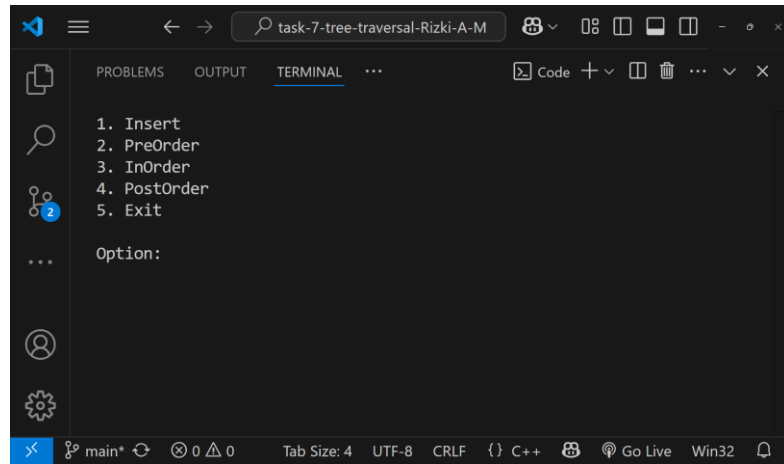
89      {
90          system("cls");
91          cout << "1. Insert\n";
92          cout << "2. PreOrder\n";
93          cout << "3. InOrder\n";
94          cout << "4. PostOrder\n";
95          cout << "5. Exit\n";
96          cout << "\nOption: "; cin >> opt;
97          switch (opt)
98          {
99
100             case 1:
101                 cout << "\n Input:";
102                 cout << "\n -----";
103                 cout << "\n New data: ";
104                 cin >> val;
105                 insert(&tree, val);
106                 break;
107
108             case 2:
109                 cout << "PreOrder Traversal\n";
110                 cout <<
111                 "=====\n";
112                 if (tree == nullptr)
113                 {
114                     cout << "Tree is empty!\n";
115                 }
116                 else
117                 {

```

```
118         preOrder(tree);
119     }
120     break;
121
122     case 3:
123         cout << "InOrder Traversal\n";
124         cout <<
125         "=====\\n";
126         if (tree == nullptr)
127         {
128             cout << "Tree is empty!\\n";
129         }
130         else
131         {
132             inOrder(tree);
133         }
134         break;
135
136     case 4:
137         cout << "PostOrder Traversal\\n";
138         cout <<
139         "=====\\n";
140         if (tree == nullptr)
141         {
142             cout << "Tree is empty!\\n";
143         }
144         else
145         {
146             postOrder(tree);
```

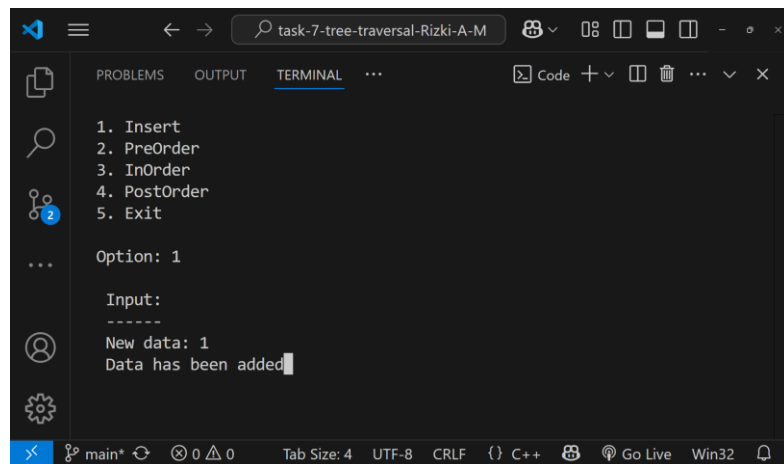
```
146         }
147         break;
148
149         case 5:
150             freeTree(tree);
151             return 0;
152
153         default:
154             cout << "Option is not valid!
Please re-enter your option";
155             break;
156     }
157     getch();
158 }
159 while(opt != 5);
160 return 0;
161 }
```

## B Output Program



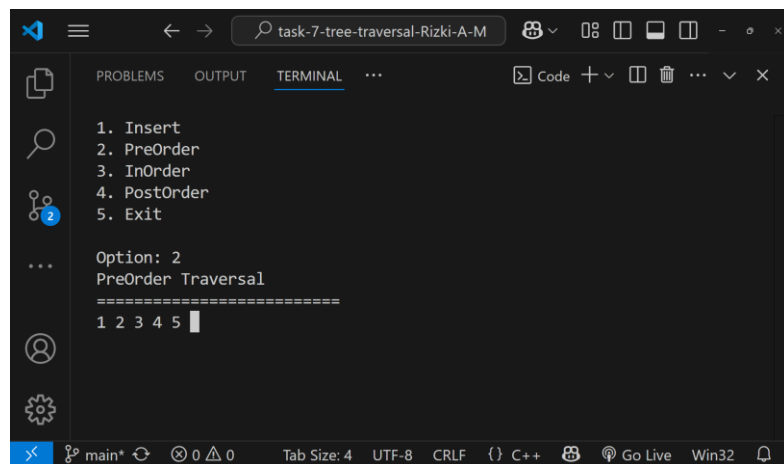
```
task-7-tree-traversal-Rizki-A-M
PROBLEMS OUTPUT TERMINAL ...
1. Insert
2. PreOrder
3. InOrder
4. PostOrder
5. Exit
Option:
```

*Gambar 2 Tampilan Menu Tree*



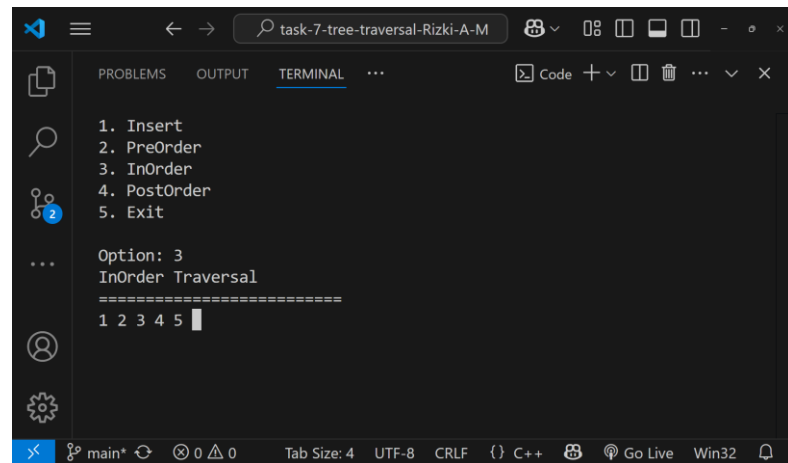
```
task-7-tree-traversal-Rizki-A-M
PROBLEMS OUTPUT TERMINAL ...
1. Insert
2. PreOrder
3. InOrder
4. PostOrder
5. Exit
Option: 1
Input:
-----
New data: 1
Data has been added
```

*Gambar 3 Tampilan Menu Insert Tree*



```
task-7-tree-traversal-Rizki-A-M
PROBLEMS OUTPUT TERMINAL ...
1. Insert
2. PreOrder
3. InOrder
4. PostOrder
5. Exit
Option: 2
PreOrder Traversal
=====
1 2 3 4 5
```

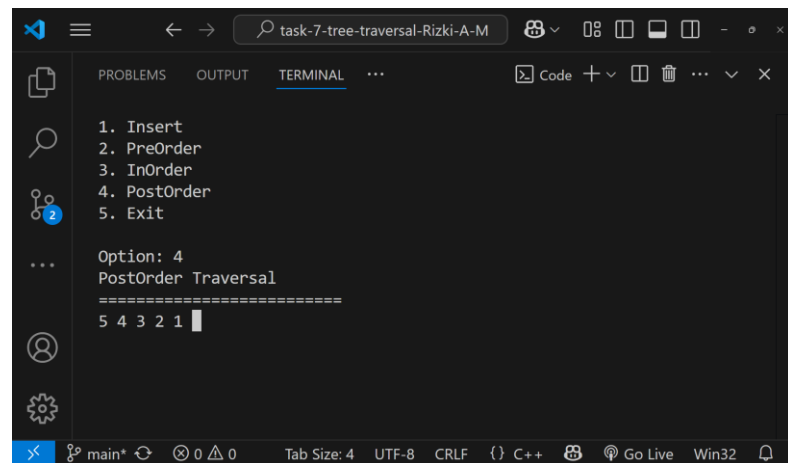
*Gambar 4 Tampilan Menu PreOrder Tree*



The screenshot shows the Visual Studio Code interface with the terminal window open. The terminal displays a menu with five options: 1. Insert, 2. PreOrder, 3. InOrder, 4. PostOrder, and 5. Exit. Below the menu, the user has entered 'Option: 3' and 'InOrder Traversal'. The terminal then displays the output '1 2 3 4 5'.

```
task-7-tree-traversal-Rizki-A-M  
PROBLEMS OUTPUT TERMINAL ...  
1. Insert  
2. PreOrder  
3. InOrder  
4. PostOrder  
5. Exit  
  
Option: 3  
InOrder Traversal  
=====
```

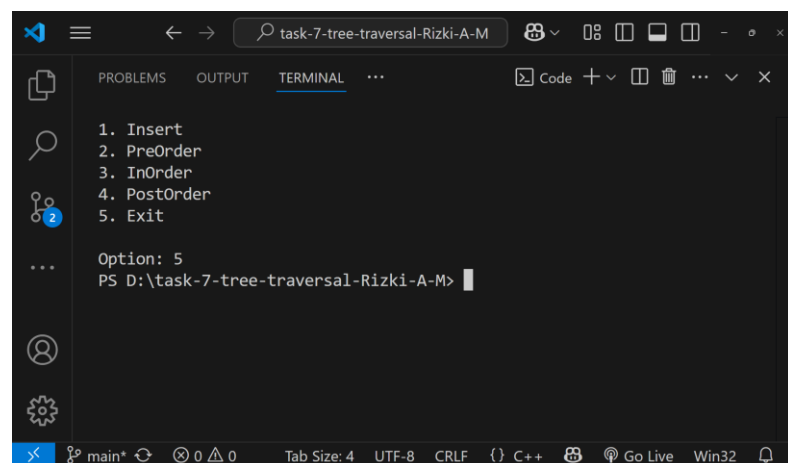
*Gambar 5 Tampilan Menu InOrder Tree*



The screenshot shows the Visual Studio Code interface with the terminal window open. The terminal displays a menu with five options: 1. Insert, 2. PreOrder, 3. InOrder, 4. PostOrder, and 5. Exit. Below the menu, the user has entered 'Option: 4' and 'PostOrder Traversal'. The terminal then displays the output '5 4 3 2 1'.

```
task-7-tree-traversal-Rizki-A-M  
PROBLEMS OUTPUT TERMINAL ...  
1. Insert  
2. PreOrder  
3. InOrder  
4. PostOrder  
5. Exit  
  
Option: 4  
PostOrder Traversal  
=====
```

*Gambar 6 Tampilan Menu PostOrder Tree*



The screenshot shows the Visual Studio Code interface with the terminal window open. The terminal displays a menu with five options: 1. Insert, 2. PreOrder, 3. InOrder, 4. PostOrder, and 5. Exit. Below the menu, the user has entered 'Option: 5'. The terminal then displays the output 'PS D:\task-7-tree-traversal-Rizki-A-M>'.

```
task-7-tree-traversal-Rizki-A-M  
PROBLEMS OUTPUT TERMINAL ...  
1. Insert  
2. PreOrder  
3. InOrder  
4. PostOrder  
5. Exit  
  
Option: 5  
PS D:\task-7-tree-traversal-Rizki-A-M>
```

*Gambar 7 Tampilan Menu Exit*



## C Pembahasan

### • Alur Program

Program ini dimulai dengan mendefinisikan *struktur Node*, yang menjadi dasar pembentuk pohon biner. Setiap *Node* terdiri dari sebuah *variabel integer* yang akan menyimpan nilai, serta dua *pointer left dan right* yang masing-masing menunjuk ke anak kiri dan anak kanan dari *node* tersebut. Pada awal program, *pointer tree* akan diinisialisasi sebagai *nullptr* yang artinya pohon biner dalam keadaan kosong.

Selanjutnya, program akan menampilkan menu utama kepada pengguna dalam sebuah *loop do while*. Menu yang ada akan menyediakan lima opsi, yaitu Insert (untuk menambahkan data baru), PreOrder (untuk menampilkan data dengan urutan *pre-order traversal*), InOrder (untuk menampilkan data dengan urutan *in-order traversal*), PostOrder (untuk menampilkan data dengan urutan *post-order traversal*), dan Exit (untuk keluar dari program). Setiap selesai menjalankan fungsi yang ada di setiap pilihan maka tampilan layar program akan dibersihkan dengan *system ("cls")* sehingga tidak adanya penumpukkan hasil output pada layar. Loop akan terus berulang selama nilai variabel pilihan tidak sama dengan 5, dan setelah selesai menjalankan program yang ada (kecuali keluar), program akan menunggu input tombol apa pun karena adanya *getch()* sebelum akhirnya akan dibersihkan oleh *system ("cls")*.

Pemilihan opsi menu diatur menggunakan switch case. Apabila pengguna memilih opsi 1 yaitu (Insert), program akan meminta input data baru yang kemudian disimpan dalam variabel *val*. Data ini lalu disisipkan ke dalam pohon melalui pemanggilan *fungsi insert()*. *Fungsi insert()* bekerja secara rekursif, yang mana apabila *root* saat ini adalah *nullptr*, *node* baru akan dibuat dan menjadi *root* di posisi tersebut. Jika data baru lebih kecil dari data pada *node root* saat ini, penyisipan akan dilanjutkan secara rekursif ke sub-pohon kiri. Sebaliknya, jika data baru lebih besar, penyisipan akan dilanjutkan ke sub-pohon kanan. Apabila data yang dimasukkan sudah ada, program akan menampilkan pesan "Data is already exist".

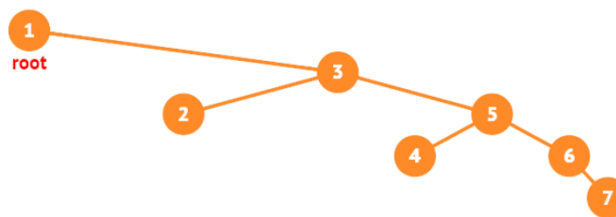
Untuk opsi 2 (PreOrder), opsi 3 (InOrder), dan opsi 4 (PostOrder), program terlebih dahulu memeriksa apakah tree kosong (*nullptr*). Apabila pohon yang ada kosong, pesan "Tree is empty!" akan ditampilkan. Kemudian, apabila pohon yang ada berisi data, *fungsi traversal* yang sesuai akan dipanggil untuk *preOrder()*, *inOrder()*, atau *postOrder()*. *Fungsi preOrder()* akan menampilkan data node saat ini, lalu mengunjungi anak kiri, dan kemudian anak kanan. *Fungsi inOrder()* akan mengunjungi anak kiri, lalu menampilkan data node saat ini, dan kemudian mengunjungi anak kanan, menghasilkan urutan data yang teratur. Sementara itu, *fungsi postOrder()* akan mengunjungi anak kiri, lalu anak kanan, dan terakhir menampilkan data node saat ini.

Terakhir, saat pengguna memilih opsi 5 (Exit), program akan memanggil *fungsi freeTree()*. Fungsi ini bertanggung jawab atas dealokasi memori yang digunakan oleh pohon biner. *freeTree()* bekerja secara rekursif dengan pendekatan *post-order traversal*, yaitu membebaskan sub-pohon kiri terlebih dahulu, lalu sub-pohon kanan, dan barulah menghapus (delete) node saat ini. Proses ini memastikan bahwa semua memori yang dialokasikan secara dinamis untuk node-node pohon dikembalikan ke sistem, mencegah *memory leak*. Setelah seluruh memori dibebaskan, program akan keluar dengan mengembalikan nilai 0. Apabila pengguna memilih opsi yang tidak valid, program akan menampilkan pesan kesalahan "Option is not valid! Please re-enter your option".

- **Insertion**

Fungsi penting untuk program ini adalah *insert()*, yang bertanggung jawab untuk menambahkan data baru ke dalam pohon. *insert()* akan diinisialisasikan dengan Binary Search Tree (BST) yaitu sebuah struktur data yang efisien untuk pencarian dan pengurutan. Fungsi ini akan mengambil dua parameter yaitu sebuah *pointer ke pointer Node* yang merepresentasikan akar (atau sub-akar) pohon saat ini, dan sebuah *integer newData* yang merupakan data yang ingin disisipkan.

Proses penyisipan dimulai dengan memeriksa apakah posisi root saat ini kosong (null). Apabila iya, berarti kita telah menemukan tempat yang tepat untuk newData. Maka, sebuah newNode akan dibuat, newData disematkan di dalamnya, dan pointer left serta right dari newNode diatur ke nullptr (karena ini adalah node daun baru). newNode ini kemudian menjadi root di posisi tersebut. Kemudian, apabila root yang ada tidak kosong, fungsi akan membandingkan newData dengan data yang ada di root saat ini. Jika newData lebih kecil, program akan memanggil insert() secara rekursif untuk subtree kiri. Sebaliknya, jika newData lebih besar, rekursi akan terjadi pada subtree kanan. Ada hal penting disini yaitu bila newData sama dengan data yang sudah ada, program akan mengeluarkan pesan bahwa data duplikat, ini berguna untuk menjaga integritas BST sesuai aturan standar.

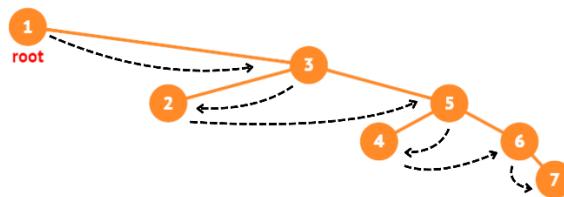


*Gambar 8 Ilustrasi Insertion Pada Tree*

- **Pre-Order Traversal**

Fungsi preOrder() berfungsi untuk melakukan traversal pada struktur data pohon (tree). Dalam metode ini, urutan kunjungan dimulai dari node induk (parent) terlebih dahulu, kemudian lanjut ke subtree kiri, dan yang terakhir ke subtree kanan. Traversal dilakukan secara rekursif, yang berarti fungsi akan terus memanggil dirinya sendiri selama masih ada node yang perlu dikunjungi. Langkah pertama dalam fungsi ini adalah memeriksa apakah node saat ini bernilai null. Apabila iya, maka tidak ada lagi node yang bisa dikunjungi dan fungsi akan berhenti.

Namun, apabila node yang ada tidak null, maka data atau nilai dari node tersebut langsung dicetak. Setelah itu, fungsi akan masuk ke subtree sebelah kiri dan melakukan proses yang sama secara rekursif. Setelah seluruh node di subtree kiri selesai dikunjungi, fungsi akan berpindah ke subtree kanan dan kembali menjalankan proses yang sama. Dengan pola tersebut, `preOrder()` selalu mengunjungi node dalam urutan: node saat ini, subtree kiri, dan subtree kanan. Pola ini berguna, misalnya, ketika ingin menyalin struktur pohon atau mengekstrak informasi dari atas ke bawah secara teratur.

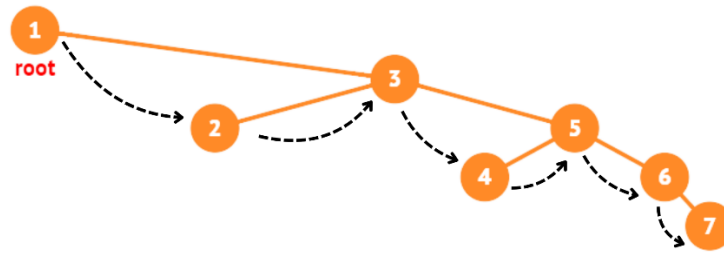


*Gambar 9 Ilustrasi PreOrder Pada Tree*

#### • In-Order Traversal

Fungsi `inOrder()` digunakan untuk melakukan penelusuran (traversal) pada struktur pohon. Pada metode ini, proses kunjungan dimulai dari subtree kiri terlebih dahulu, kemudian ke node induk (parent), dan terakhir ke subtree kanan secara rekursif. Urutan ini sangat berguna karena pada tree bertipe binary search tree (BST), traversal in-order akan mencetak data dalam urutan yang terurut dari yang terkecil hingga terbesar.

Saat fungsi dijalankan, hal pertama yang dilakukan adalah memeriksa apakah node saat ini bernilai null. Apabila null, berarti tidak ada node lagi yang bisa dikunjungi, sehingga fungsi akan berhenti. Namun, apabila terdapat node, maka fungsi akan lebih dulu mengunjungi subtree kiri secara rekursif. Setelah itu, data dari node induk akan dicetak. Terakhir, fungsi akan mengunjungi subtree kanan dan mencetak datanya. Pola kunjungannya mengikuti urutan dari kiri ke induk dan ke kanan, dan ini berulang hingga seluruh node telah dikunjungi.

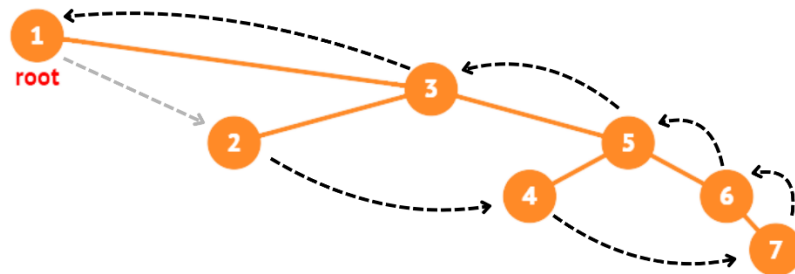


*Gambar 10 Ilustrasi InOrde Pada Tree*

### • Post-Order Traversal

Fungsi `postOrder()` digunakan untuk melakukan traversal pada struktur pohon. Dalam metode ini, urutan kunjungannya dimulai dari subtree kiri, lalu ke subtree kanan, dan terakhir ke node induk (parent) secara rekursif, dan cocok digunakan ketika kita ingin memproses atau menghapus node setelah semua turunannya selesai dikunjungi, seperti saat menghapus seluruh isi pohon.

Cara kerjanya dimulai dengan memeriksa apakah node saat ini bernilai null. Apabila null, maka tidak ada node lagi yang bisa dikunjungi, maka fungsi akan berhenti. Namun apabila node ada, maka program pertama-tama akan mengunjungi subtree kiri secara rekursif. Setelah itu, program lanjut ke subtree kanan dan melakukan hal yang sama. Jika kedua sisi sudah selesai dikunjungi, barulah program mencetak data dari node induk. Dengan demikian, urutan traversal-nya adalah dari kiri ke kanan dan akhirnya ke induk, dan pola ini berulang hingga seluruh node dalam tree selesai dikunjungi.



*Gambar 11 ilustrasi PostOrder Pada Tree*

### **TAUTAN GITHUB**

[https://github.com/Rizki-A-M/Rizki-A-M-PRAKTIKUM\\_ALGORITMA\\_DAN\\_STRUKTUR\\_DATA.git](https://github.com/Rizki-A-M/Rizki-A-M-PRAKTIKUM_ALGORITMA_DAN_STRUKTUR_DATA.git)