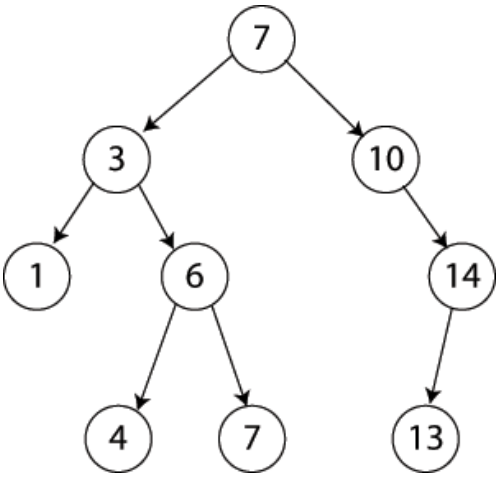
PROGRAM STUDI INFORMATIKA
Universitas Syiah Kuala

Struktur Data dan Algoritma

Binary Search Tree (BST)

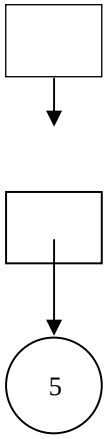
Dr. Taufik Fuadi Abidin, M.Tech
tfa@informatika.unsyiah.ac.id



Binary Tree

Binary Tree adalah struktur data yang mirip dengan *Linked List*. Bila *Linked List* dianalogikan sebagai rantai yang linier maka *Binary Tree* dianalogikan sebagai pohon. *Binary Tree* dikelompokkan menjadi tree yang tidak berurut (*unordered Binary Tree*) dan tree yang terurut (*ordered Binary Tree*).

Binary Tree dapat digambarkan berdasarkan kondisinya, yaitu:

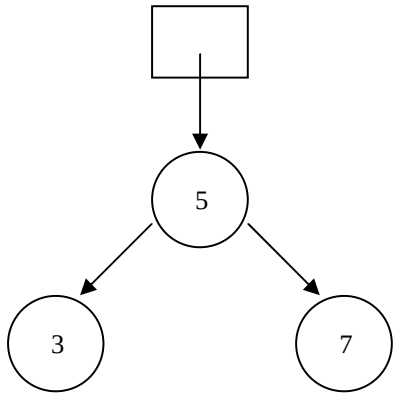


Binary Tree kosong

Pointer ke akar (*root*) dari tree

Binary Tree sebagai *root* sekaligus sebagai daun (*leaf*)

Gambaran dari *Binary Tree* yang terdiri dari 3 (tiga) *node*:



Pointer ke root dari tree

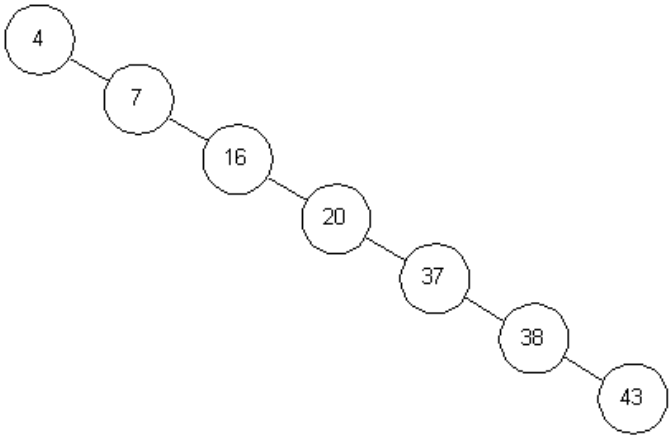
Root dari tree

Leaf dari tree

Binary Search Tree

Binary Search Tree (BST) merupakan tree yang terurut (*ordered Binary Tree*) yang memiliki kelebihan bila dibanding dengan struktur data lain. Diantaranya adalah proses pengurutan (*sorting*) dan pencarian (*searching*) dapat dilakukan bila data sudah tersusun dalam struktur data BST. Pengurutan dapat dilakukan bila BST ditelusuri (*traversed*) menggunakan metode in-order. Detail dari proses penelusuran ini akan dibahas pada pertemuan selanjutnya. Data yang telah tersusun dalam

struktur data BST juga dapat dicari dengan mudah dan memiliki rata-rata kompleksitas sebesar *O(log n)*, namun membutuhkan waktu sebesar O(n) pada kondisi terjelek dimana BST tidak berimbang dan membentuk seperti linked list seperti contoh berikut:

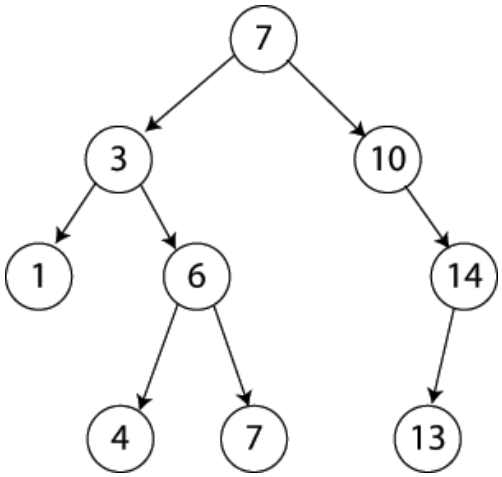


Aturan Dalam Membangun BST

Agar data benar-benar tersusun dalam struktur data BST, dua aturan yang harus dipenuhi pada saat data diatur dalam BST adalah sebagai berikut:

1. Semua data dibagian kiri *sub-tree* dari *node t* selalu lebih kecil dari data dalam *node t* itu sendiri.
2. Semua data dibagian kanan *sub-tree* dari *node t* selalu lebih besar atau sama dengan data dalam *node t*.

BST berikut adalah sebuah BST berukuran 9 dengan kedalaman 3 dengan node daun (leaf) adalah 1, 4, 7 dan 13.



Contoh Aplikasi BST

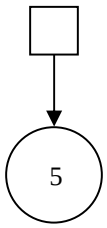
1. Membangun daftar vocabulary yang merupakan bagian dari inverted index (sebuah struktur data yang digunakan oleh banyak mesin pencari seperti Google.com, Yahoo.com dan Ask.com)

2. Banyak digunakan dalam bahasa pemrograman untuk mengelola dan membangun *dynamic sets*.

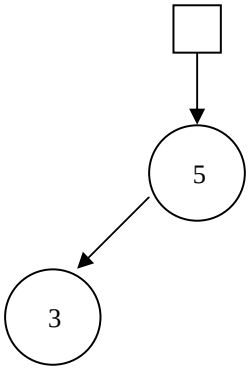
Pembentukan BST

Bila diketahui sederetan data 5, 3, 7, 1, 4, 6, 8, 9 maka pemasukan data tersebut dalam struktur data BST, langkah per langkah, adalah sebagai berikut:

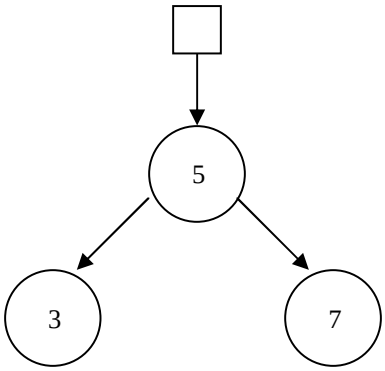
Langkah 1: Pemasukan data 5 sebagai root



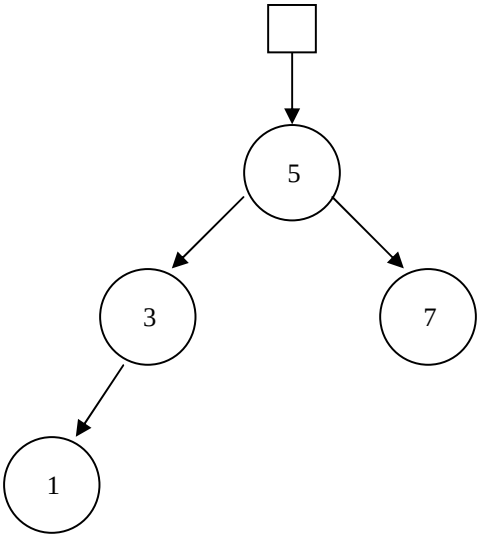
Langkah 2: Pemasukan data 3 disebelah kiri simpul 5 karena 3 < 5.



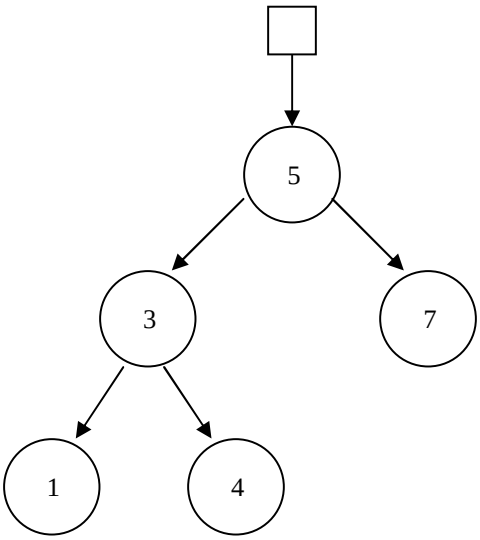
Langkah 3: Pemasukan data 7 disebelah kanan simpul 5 karena 7 > 5.



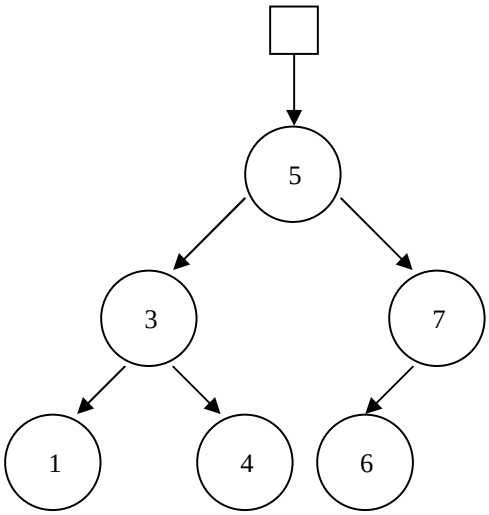
Langkah 4: Pemasukan data 1. Karena data 1 lebih kecil dari data di root yaitu 5 maka penelusuran dilanjutkan kesebelah kiri root. Kemudian karena disebelah kiri sudah ada daun dengan nilai 3 dan data 1 lebih kecil dari data 3 maka data 1 disisipkan disebelah kiri simpul 3.



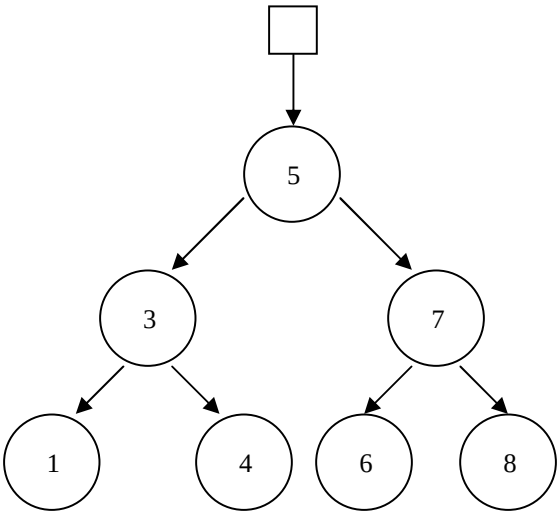
Langkah 5: Pemasukan data 4.



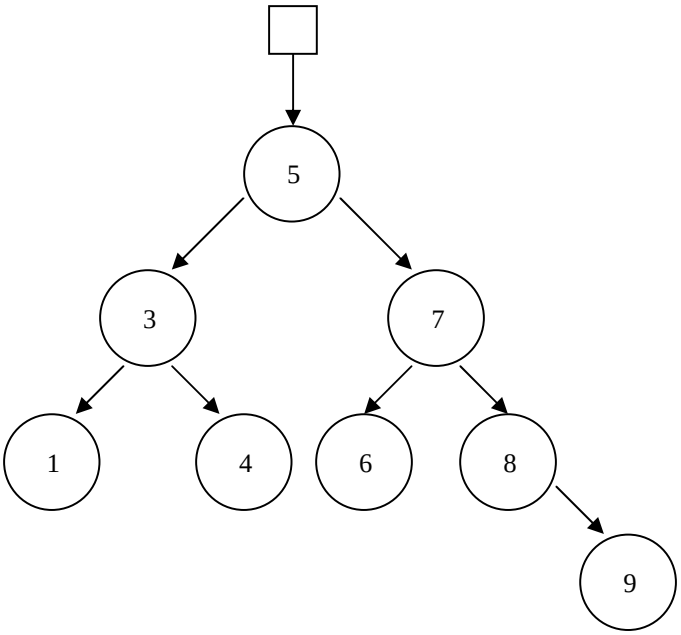
Langkah 6: Pemasukan data 6. Karena data 6 lebih besar dari data di root yaitu 5 maka penelusuran dilanjutkan kesebelah kanan root. Kemudian karena disebelah kanan sudah ada simpul dengan nilai 7 dan data 6 lebih kecil dari data 7 maka data 6 disisipkan disebelah kiri simpul 7.



Langkah 7: Pemasukan data 8. Karena data 8 lebih besar dari data di root yaitu 5 maka penelusuran dilanjutkan kesebelah kanan root. Kemudian karena disebelah kanan sudah ada simpul dengan nilai 7 dan karena data 8 lebih besar dari data 7 maka data 8 disisipkan disebelah kanan simpul 7.



Langkah 8: Pemasukan data 9. Karena data 9 lebih besar dari data di root yaitu 5 maka penelusuran dilanjutkan kesebelah kanan root. Kemudian karena disebelah kanan bukan merupakan daun yaitu simpul dengan nilai 7 dan karena data 9 lebih besar dari data 7 penelusuran terus dilanjutkan kesebelah kanan. Selanjutnya ditemukan daun dengan nilai 8, karena data 9 lebih besar dari 8 maka data 9 disisipkan disebelah kanan simpul 8.



Contoh Implementasi BST Menggunakan Bahasa C

Diskusikan secara kelompok implementasi dari algoritma *Binary Search Tree* berikut ini.

```
/* file bst.h */

typedef struct intBSTNode * IntBSTNodePtr;
typedef struct intBSTNode {
    int data;
    IntBSTNodePtr left, right;
} IntBSTNode;

typedef struct root {
    IntBSTNodePtr    root;
    unsigned     size;
} IntBSTree;

int MakeIntBST(IntBSTree *);
int InsertBST(IntBSTree *, int);
void freeBST(IntBSTNode *);
void inOrder(IntBSTNode *);
```

Bagian Deklarasi

Bagian deklarasi di atas diasumsikan disimpan dalam sebuah file *header* dengan nama *bst.h.* Fungsi-fungsi di bawah ini diasumsikan disimpan dalam file dengan nama *bst.c*

```
/* file bst.c */

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

int MakeIntBST(IntBSTree * pBST) {
    pBST->root = NULL;
    pBST->size = 0;
    return EXIT_SUCCESS;
}
```

```
int InsertBST(IntBSTree * pBST, int data) {
    IntBSTNodePtr current, previous, new;
    previous = NULL;
    current = pBST->root;
    while(current != NULL) {
        previous = current;
        if (data < current->data) {
            current = current->left;
        }
        else {
            current = current->right;
        }
    }
    new = malloc(sizeof(IntBSTNode));
    if (new == NULL) {
        return EXIT_FAILURE;
    }
    new->data = data;
    new->left = NULL;
    new->right = NULL;
    (pBST->size)++;

    if (previous == NULL) {
        pBST->root = new;
        return EXIT_SUCCESS;
    }

    if (data < previous->data) {
        previous->left = new;
    }
    else {
        previous->right = new;
    }
    return EXIT_SUCCESS;
}
```

```
void freeBST(IntBSTNode * pBST)
{
   if(pBST!=NULL) {
      freeBST(pBST->left);
      free(pBST);
      freeBST(pBST->right);
     free(pBST);
   }
}

void inOrder(IntBSTNode * pBST)
{
   if(pBST!=NULL)
   {
      inOrder(pBST->left);
      printf("%d->",pBST->data);
      inOrder(pBST->right);
   }
   printf("\n");
}


/* Program utama */

int main(void) {
     IntBSTree ibst;
     MakeIntBST(&ibst);

     5, 3, 7, 1, 4, 6, 8, 9
     InsertBST(&ibst,5);
     InsertBST(&ibst,3);
     InsertBST(&ibst,7);
     InsertBST(&ibst,1);
     InsertBST(&ibst,4);
     InsertBST(&ibst,6);
     InsertBST(&ibst,8);
     InsertBST(&ibst,9);

     inOrder(ibst.root);

     freeBST(ibst.root);
     return EXIT_SUCCESS;
}
```