1. Please determine whether these trees are Height-Balanced Binary Search Tree.

|  |  |
| --- | --- |
| a. | b. |
| c. | d. |

1. Given an array a[10]={3,2,1,4,5,6,7,10,9,8}, answer the following questions.
2. Build a corresponding Binary Search Tree from index 0, and find the node with the largest balance factor(i.e. the absolute value of the difference of the height of the left subtree and the height of the right subtree).
3. Build an AVL Tree of the array from index 0, and list all the tree building steps.
4. Given a partial AVL Tree code as follows, please answer the following questions.

class BiTNode{

    private:

        int data;

        int bf;    //bf=balanced factor

        BiTNode \*lchild,\*rchild;

    public:

        BiTNode(); //Skip to define constructor

        int getBF(){

            return bf;

        }

        BiTNode\* getLeft(){

            return this->lchild;

        }

        BiTNode\* getRight(){

            return this->rchild;

        }

        void setLeft(BiTNode \*node){

            this->lchild=node;

        }

        void setRight(BiTNode \*node){

            this->rchild=node;

        }

};

class AVLTree{

    private:

        BiTNode \*nodeArray;

    public:

        AVLTree(); // Skip to define constructor

        void R\_Rotate(BiTNode \*\*P);

        void L\_Rotate(BiTNode \*\*P);

        void LeftBalance(BiTNode \*\*T);

}

1. Please explain R\_Rotate(BiTNode \*P).

void AVLTree::R\_Rotate(BiTNode \*\*P){

    BiTNode \*L;

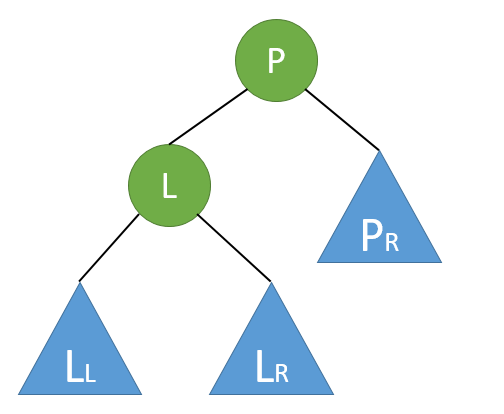
    L=(\*P)->getLeft();

    (\*P)->setLeft(L->getLeft());

    L->setRight(\*\*P);

    \*P=L;

}



1. Design L\_Rotate(BiTNode \*\*P).
2. Following the previous question, please complete the LeftBalaced(BiTNode \*\*T)

#define LH +1  // Left height

#define EH 0   // Equal height

#define RH -1  // Right height

/\*對以指標T所指節點為根的二元樹做左旋轉平衡處理\*/

/\*本演算法結束時，指標T指向新的根節點\*/

void AVLTree::LeftBalance(BiTNode \*\*T){

    BiTNode\* L,Lr;

    L=(\*T)->getLeft;

    switch(L->getBF()){

        /\* 檢查T的左子樹平衡度，並做相應的平衡處理 \*/

        case LH:  /\* 新節點插入在T的左孩子的左子樹上，要做右旋轉處理 \*/

            // TODO(1)

        case RH:  /\* 新節點插入在T的左孩子的右子樹上，要做雙旋轉處理 \*/

            // TODO(2)

    }

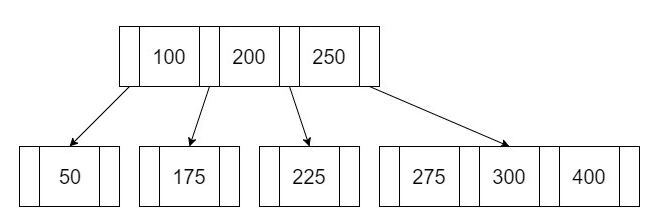
}

1. Complete TODO(1)
2. Complete TODO(2)
3. Can any node in an AVL Tree has the balanced factor equal to ±3? Why?
4. (1) Explain the concept of the red-black tree. What are the differences between the red-black tree and the AVL tree.

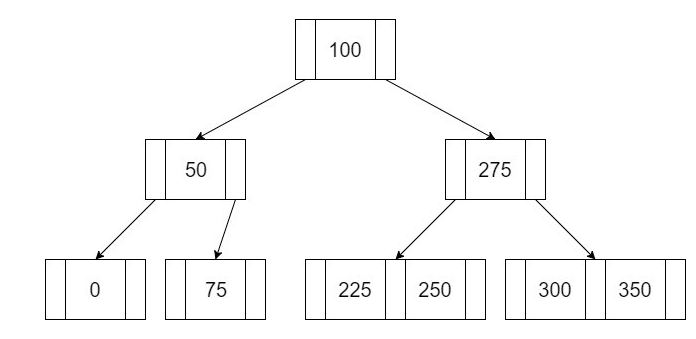
(2) Describe the rules of insertion and deletion.

(3) Build a red-black tree following the sequence 10,20,30,15

1. How to handle duplicated keys in an AVL tree. Describe your method.
2. The minimal degree of the following B tree is 2. Please insert a new key “350” into the B tree. Please explain the process step by step.



1. The minimal degree of the following B tree is 2. Please delete the key “75” in the B tree. You need to explain the process step by step.



1. Present an interesting or useful data structure that has not been discussed in this class.