

Department of Computer Science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

Trees

Binary Search Tree (BST): Deletion

Dr. Shylaja S S
Ms. Kusuma K V

Deletion of a Node in Binary Search Tree

Consider the following 3 cases for deletion of a node in Binary Search Tree so that even after the node is deleted the BST property is preserved.

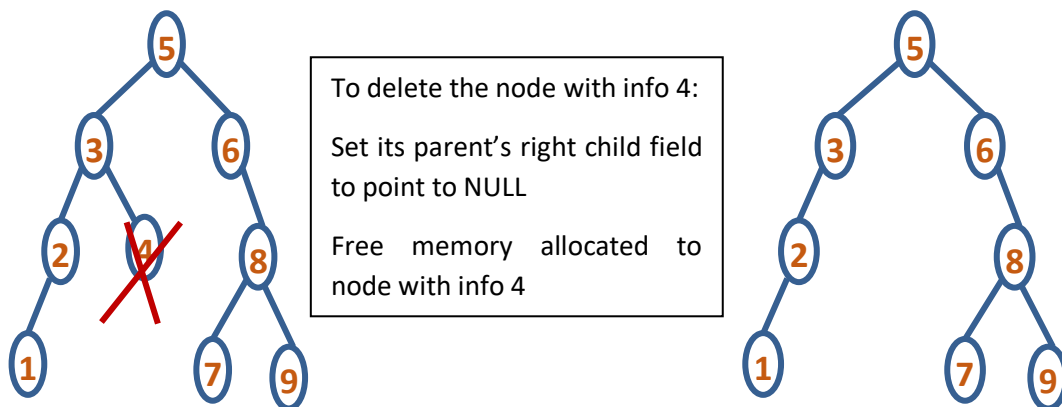
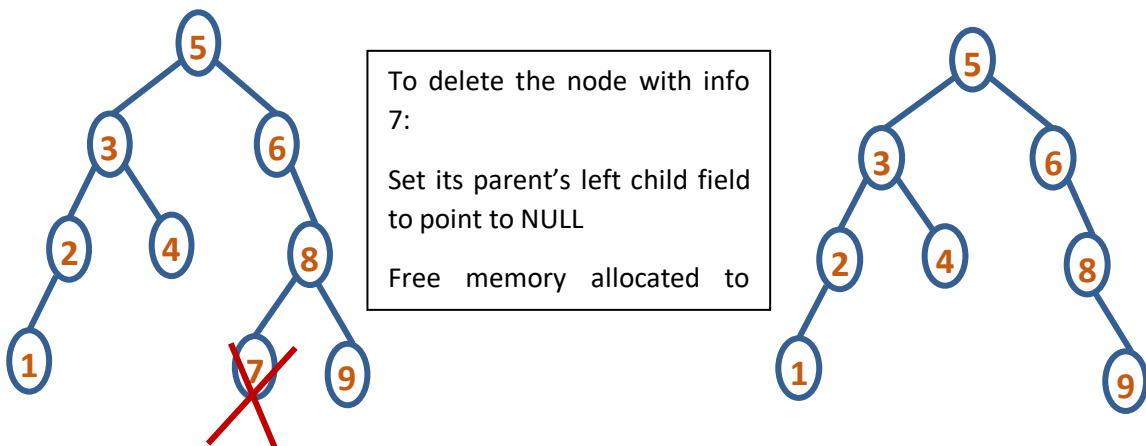
Case 1: Node with no child (leaf node)

Case 2: Node with 1 child

Case 3: Node with 2 children

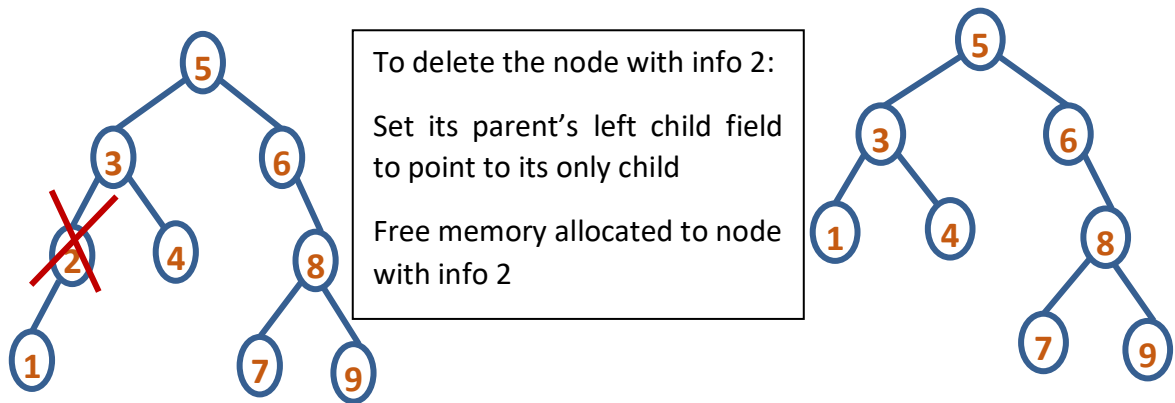
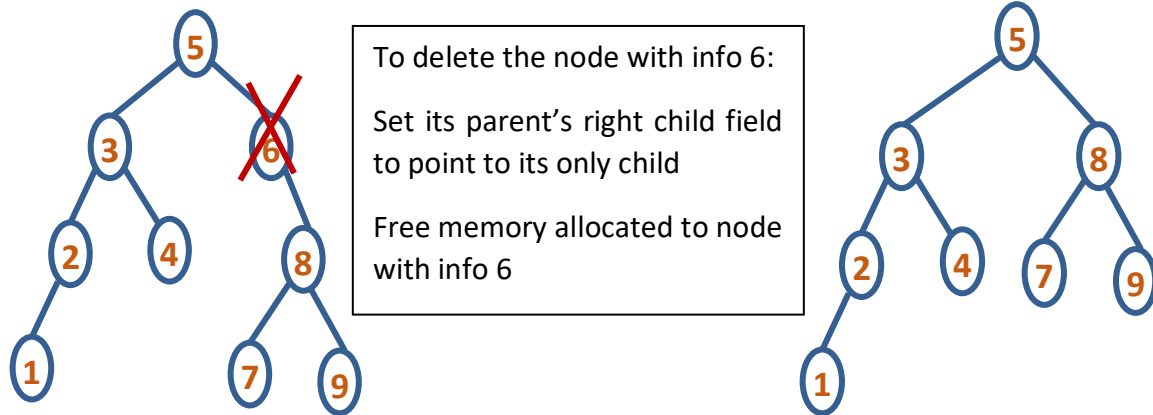
Case 1: Deletion of a leaf node

Set its parent's corresponding child field to point to NULL. Free memory allocated to the node.



Case 2: Deletion of a node with one child

Connect the node's parent with node's child node



Case 3: Deletion of a node with 2 children

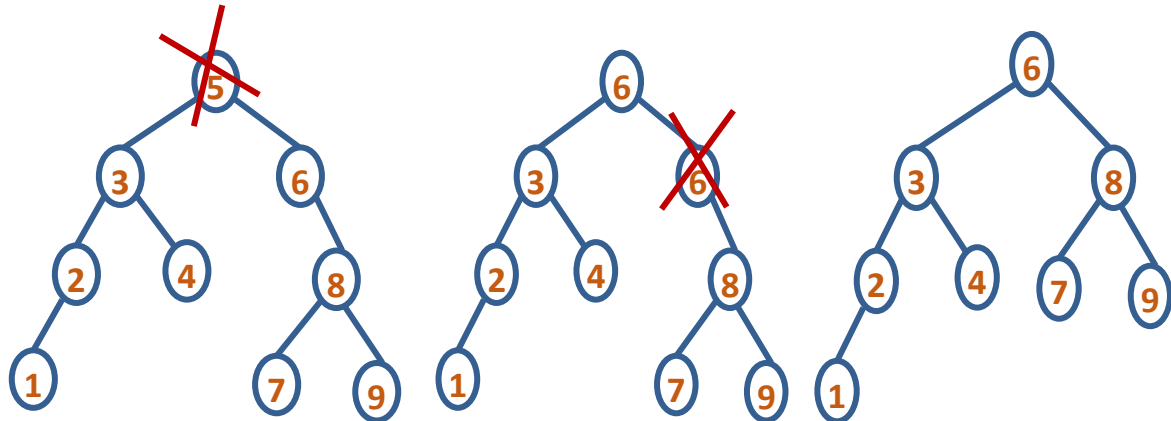
This case can be handled in 2 ways:

Way 1: Replace the node to be deleted with its inorder successor (Smallest in its Right subtree or Leftmost in its Right subtree) and delete that inorder successor.

Way2: Replace the node to be deleted with its inorder predecessor (Largest in its Left subtree) and delete that inorder predecessor.

For the deletion of inorder successor/predecessor, case 3 gets reduced to either case 1 or case 2. We know how to solve case1 and case2.

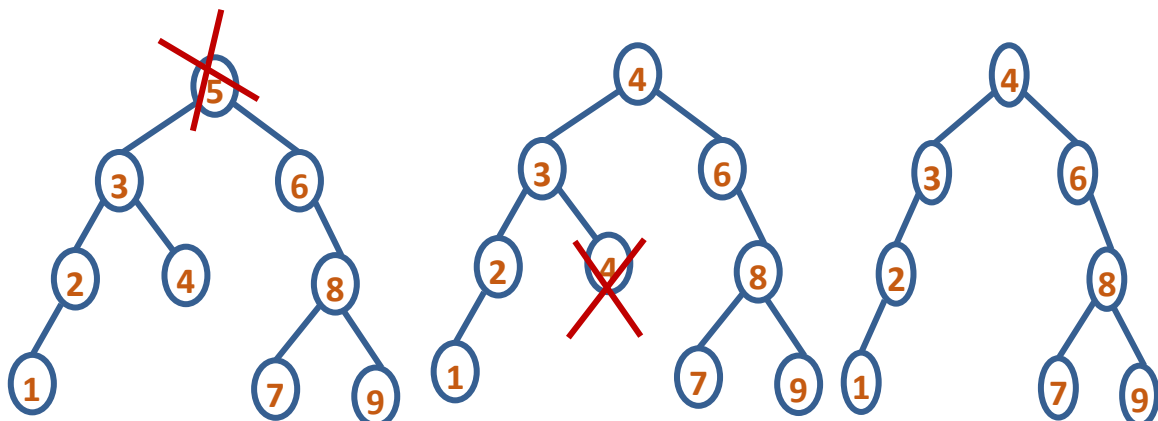
To delete node with info 5 (Way 1: Replace with inorder successor)



Replace 5 with its
inorder successor

Now delete that inorder successor
Now case3 has got changed to
case2 (In general may change to
case2 or case1)

To delete node with info 5 (Way 2: Replace with inorder predecessor)



Replace 5 with its
inorder predecessor

Now delete that inorder predecessor
Now case3 has got changed to case1
(In general may change to case2 or
case1)

Note: For implementation we shall consider replacing with inorder successor.

//BST Deletion

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
    int info;
    struct node *left,*right;
}NODE;
typedef struct tree
{
    NODE *root;
}TREE;

void init(TREE *pt)
{
    pt->root=NULL;
}

NODE* createNode(int e)
{
    NODE *temp=malloc(sizeof(NODE));
    temp->left=NULL;
    temp->right=NULL;
    temp->info=e;
    return temp;
}

void create(TREE *pt)
{
    NODE *p,*q;
    int e, wish;

    printf("Enter info\n");
    scanf("%d",&e);

    pt->root=createNode(e);

    do{
        printf("Enter info\n");
        scanf("%d",&e);

        q=NULL;
```

```
p=pt->root;

while(p!=NULL)
{
    q=p;

    if(e < p->info)
        p = p->left;
    else
        p = p->right;
}

if(e < q->info)
    q->left = createNode(e);
else
    q->right = createNode(e);

printf("Do you wish to continue\n");
scanf("%d",&wish);
}while(wish);
}

void io(NODE* r)
{
    if(r!=NULL)
    {
        io(r->left);
        printf("%d ",r->info);
        io(r->right);
    }
}

void inorder(TREE *pt)
{
    io(pt->root);
}
```

```

NODE* delNode(NODE *r,int ele)
{
    NODE *temp,*p;

    if(r==NULL)
        return r;

    if(ele < r->info)
        r->left = delNode(r->left,ele);
    else if(ele > r->info)
        r->right = delNode(r->right,ele);
    else
    {
        if(r->left == NULL)                //1 right child or No children
        {
            temp=r->right;
            free(r);
            return temp;
        }
        else if(r->right == NULL)          //1 left child or No children
        {
            temp=r->left;
            free(r);
            return temp;
        }
        else
        {
            //Node to be deleted has both children
            //Finding p's leftmost child which is the inorder successor
            p=r->right;
            while(p->left != NULL)
                p=p->left;

            r->info=p->info;
            r->right=delNode(r->right, p->info);
        }
    }
    return r;
}

void deleteNode(TREE *pt,int e)
{
    pt->root=delNode(pt->root,e);
}

```

```
int main()
{
    int e;
    TREE t;
    init(&t);
    create(&t);
    inorder(&t);
    printf("Enter the element to be deleted\n");
    scanf("%d",&e);
    deleteNode(&t,e);
    printf("After deletion\n");
    inorder(&t);

    return 0;
}
```