```
#include <stdio.h>
#include <stdlib.h>


typedef struct node{
   int data;
   int bf;
   struct node *lchild;
   struct node *rchild;
}NODE;

int max(int a,int b){
   if(a>b)
      return a;
   return b;
}

int height(NODE *root){
   if(root==NULL)
      return 0;
   int lheight=height(root->lchild);
   int rheight=height(root->rchild);
   return max(lheight,rheight)+1;
}

void updateBF(NODE *root){

root->bf=height(root->lchild)-height(root->rchil
d);
}

NODE* leftrotate(NODE* root){
   NODE* pptr=root;
   NODE* aptr=pptr->rchild;
   pptr->rchild=aptr->lchild;
   aptr->lchild=pptr;
   updateBF(pptr);updateBF(aptr);
   return aptr;
}

NODE* rightrotate(NODE* root){
   NODE* pptr=root;
   NODE* aptr=pptr->lchild;
   pptr->lchild=aptr->rchild;
   aptr->rchild=pptr;
   updateBF(pptr);updateBF(aptr);
   return aptr;
}

NODE* leftrightrotate(NODE *root){
   NODE*x=root,*y=root->lchild;
   y=leftrotate(y);
   x->lchild=y;
   x=rightrotate(x);
   return x;
}

NODE* rightleftrotate(NODE *root){
   NODE*x=root,*y=root->lchild;
   y=rightrotate(y);
   x->lchild=y;
   x=leftrotate(x);
   return x;
}

NODE* rotations(NODE* root){
   if(root->bf==2&&root->lchild->bf==1){
      //printf("\nrightrotate\n");
      root=rightrotate(root);
      return root;
   }
   if(root->bf==-2&& root->rchild->bf==-1){
      //printf("\nleftrotate\n");
      root=leftrotate(root);
      return root;
   }
   if(root->bf==2&&root->lchild->bf==-1){
      //printf("\nleftrightrotate\n");
      root=leftrightrotate(root);
      return root;
   }
   if(root->bf==-2&&root->rchild->bf==1){
      //printf("\nrightleftrotate\n");
      root=rightleftrotate(root);
      return root;
   }
   return root;
}

NODE *delete(NODE *root,int dkey){

   if(root==NULL){
      printf("key not prestent");
      return root;
   }
   if(dkey<root->data){
      root->lchild=delete(root->lchild,dkey);
   }
   else if(dkey>root->data){
      root->rchild=delete(root->rchild,dkey);
   }
   else{

      if(root->lchild!=NULL &&
root->rchild!=NULL){
         NODE *suc=root->rchild;
         while(suc->lchild!=NULL){
            suc=suc->lchild;
         }
         root->data=suc->data;

root->rchild=delete(root->rchild,suc->data);
      }
      else{

         NODE *temp=root;
         if(root->lchild!=NULL)
            root=root->lchild;
         else if(root->rchild!=NULL)
            root=root->rchild;
            else
                root=NULL;

            free(temp);

        }

    }
        if(root!=NULL){
            updateBF(root);

        if(abs(root->bf)>1){
            printf("rotate\n");
            printf("%d\n",root->data);
            root=rotations(root);
        }
    }

    return root;
}

NODE* insert(NODE *root,int data){
    if(root==NULL){
        root=(NODE*) malloc(sizeof(NODE));
        root->data=data;
        root->bf=0;
        root->lchild=NULL;
        root->rchild=NULL;
        return root;
    }
    if(data<root->data){
        root->lchild=insert(root->lchild,data);
    }
    if(data>root->data){
        root->rchild=insert(root->rchild,data);

    }
    updateBF(root);
    if(abs(root->bf)>1){
        root=rotations(root);

    }
    return root;
}

void search(NODE *root,int data){
    if(root==NULL){
        printf("data not present\n");
        return;
    }
    if(data<root->data){
        search(root->lchild,data);
    }
    else if(data>root->data){
        search(root->rchild,data);

    }
    else{
        printf("data present\n");
    }
    return;
}

void display_inorder(NODE *root){
    if(root==NULL)
        return;
    display_inorder(root->lchild);
    printf("%d:%d ",root->data,root->bf);
    display_inorder(root->rchild);
}

void minmax(NODE*root){
    NODE *tmp=root;
    while(tmp->lchild!=NULL)
        tmp=tmp->lchild;
    printf("Min value: %d ",tmp->data);
    while(root->rchild!=NULL)
        root=root->rchild;
    printf("Max value: %d\n",root->data);
}

int width(NODE *root){
    if(root==NULL)
        return 0;
    int lheight=height(root->lchild);
    int rheight=height(root->rchild);
    int lwidth=width(root->lchild);
    int rwidth=width(root->rchild);
    return
max(lheight+rheight+1,max(lwidth,rwidth));
}

void display_decend(NODE *root){
    if(root==NULL)
        return;
    display_decend(root->rchild);
    printf("%d:%d ",root->data,root->bf);
    display_decend(root->lchild);
}

int main(){
    NODE *root=NULL;
    int data;
    int m=1;
    while(m){
        printf("1:Insert a node\n2:Delete a
node\n3:Display Inorder\n4:Search\n5:Max
and Min value\n6:Display in decending
order\n7: Width of the tree\n");
        scanf("%d",&m);
        switch(m){
            case 1:
                printf("Enter Data to be inserted\n");
                scanf("%d",&data);
                root=insert(root,data);
                break;
            case 2:
                printf("Enter Data to be deleted\n");
                scanf("%d",&data);
                root=delete(root,data);
                break;
            case 3:
                printf("Inorder traverse: ");
                display_inorder(root);
                printf("\n");
                break;
           case 4:
                printf("Enter data to be searched\n");
                scanf("%d",&data);
                search(root,data);
                break;
            case 5:
                minmax(root);
                break;
            case 6:
                printf("Decending order\n");
                display_decend(root);
                printf("\n");
                break;
            case 7:
                printf("width is %d\n",width(root));
                break;
            default:
                break;
        }
    }

}
```

O/P :

```
/tmp/dJnQTFLauf.o
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
1
Enter Data to be inserted
4
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
1
Enter Data to be inserted
3
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
4
Enter data to be searched
2
data not present
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
1
Enter Data to be inserted
5
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
1
Enter Data to be inserted
6
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
1
Enter Data to be inserted
11
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
2
Enter Data to be deleted
11
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
3
Inorder traverse: 3:0 4:-1 5:0 6:1
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
5
```

```
Min value: 3 Max value: 6
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
7
width is 4
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
6
Decending order
6:1 5:0 4:-1 3:0
1:Insert a node
2:Delete a node
3:Display Inorder
4:Search
5:Max and Min value
6:Display in decending order
7: Width of the tree
```