NAME-YASH GANDHI SE IT BATCH A 14

KD Tree:

OUTPUT-

```
spit@DB-Lab-406-U15:~/Desktop$ gcc kdtree.c
spit@DB-Lab-406-U15:~/Desktop$ ./a.out
Enter the number of pts to be inserted in the tree
7
Give Coordinates for Node 1
3
4
Give Coordinates for Node 2
3
5
Give Coordinates for Node 3
3
6
Give Coordinates for Node 4
2
4
Give Coordinates for Node 5
9
8
Give Coordinates for Node 6
7
5
Give Coordinates for Node 7
0
2
(3,4)   (2,4)   (0,2)   (3,5)   (3,6)   (9,8)   (7,5)
Which pt do you want search....?
4
Which pt do you want search....?
5
The pt is absent...
Delete which coordinate
3
Delete which coordinate
4
(3,5)   (2,4)   (0,2)   (3,6)   (7,5)   (9,8)   spit@DB-Lab-406-U15:~/Desktop$
```

```
#include<stdio.h>

#include<stdlib.h>


typedef struct Node

{

        int pt[2];

        struct Node *left;

        struct Node*right;

}Node;
```

```
Node* get_Node(int a[])  //creating a node for the coordinates
{

        Node *temp = malloc(sizeof(Node));
        for(int i=0;i<2;i++)
        {
                temp->pt[i]=a[i];
        }
        temp->left=temp->right=NULL;
        return temp;
}


Node* insertNode(Node *root,int depth,int pt[])//INSERT NODE
{

        if (root == NULL)
    return get_Node(pt);
        int current_dim = depth % 2;

        if (pt[current_dim] < (root->pt[current_dim]))
     root->left  = insertNode(root->left, depth + 1, pt);
        else
```

```
        root->right = insertNode(root->right, depth + 1, pt);
            return root;
}



Node* insertroot(Node *root,int pt[])
{

            return insertNode(root, 0 , pt);
}



Node *minNode(Node *x, Node *y, Node *z, int d) //find the minimum
{
   Node *res = x;
   if (y != NULL && y->pt[d] < res->pt[d])
      res = y;
   if (z != NULL && z->pt[d] < res->pt[d])
      res = z;
   return res;
}



Node *findMin(Node* root, int depth, int d)
{
```

```
    if (root == NULL)
        return NULL;



    int curr_dim = depth % 2;



    if (curr_dim == d)
    {
        if (root->left == NULL)
            return root;
        return findMin(root->left, d, depth+1);
    }

    return minNode(root,findMin(root->left, d, depth+1), findMin(root->right, d, depth+1), d);
}



Node* find(Node* root, int d)
{

    return findMin(root, d, 0);
```

```
}



void copypt(int p1[], int p2[])
{
            int  i;
   for (i=0; i<2; i++)
       p1[i] = p2[i];
}


  int arePointsSame(int point1[], int point2[])
{
            int i;
    // Compare individual point values
    for (i = 0; i < 2; ++i)
        if (point1[i] != point2[i])
            return 0;


    return 1;
}


Node *deleteNode(Node *root, int pt[], int depth)
{
    if (root == NULL)
        return NULL;
```

```
int curr_dim = depth % 2;



if (arePointsSame(root->pt, pt))
{

   if (root->right != NULL)
   {

      Node *min = find(root->right, curr_dim);



      copypt(root->pt, min->pt);



      root->right = deleteNode(root->right, min->pt, depth+1);
   }
   else if (root->left != NULL)
   {
      Node *min = find(root->left, curr_dim);
      copypt(root->pt, min->pt);
      root->right = deleteNode(root->left, min->pt, depth+1);
   }
   else
```

```
        {
            free(root);
            return NULL;
        }
        return root;
    }



    if (pt[curr_dim] < root->pt[curr_dim])
        root->left = deleteNode(root->left, pt, depth+1);
    else
        root->right = deleteNode(root->right, pt, depth+1);
    return root;
}



Node* delete(Node *root, int pt[])
{


   return deleteNode(root, pt, 0);
}



void preOrder(Node *root)
{
```

```
        if(root==NULL)
                return;
        printf("(%d,%d)\t",root->pt[0],root->pt[1]);
        preOrder(root->left);
        preOrder(root->right);
}




int search_pt(Node* root, int pt[], int depth)
{

    if (root == NULL)
        return 0;
    if (arePointsSame(root->pt, pt))
        return 1;



    int curr_depth = depth % 2;



    if (pt[curr_depth] < root->pt[curr_depth])
```

```
        return search_pt(root->left, pt, depth + 1);


    return search_pt(root->right, pt, depth + 1);
}



int search(Node* root, int pt[])
{


    return search_pt(root, pt, 0);
}


int main()
{
            Node *root = NULL;
            int pts[20][2];
            printf("Enter the number of pts to be inserted in the tree\n");
            int n,i,j;
            scanf("%d",&n);


            for (i=0; i<n; i++)
            {
            printf("Give Coordinates for Node %d\n",i+1);
                        for(j=0;j<2;j++)
                        {
```

```
                    scanf("%d",&pts[i][j]);
                }
}

for (i=0; i<n; i++)
root = insertroot(root, pts[i]);
preOrder(root);
printf("\n");
int pt1[2],pt2[2];
for (i=0; i<2; i++)
{
                printf("Which pt do you want search....?\n");
                scanf("%d",&pt1[i]);
}

int x=search(root, pt1);

if(x==1)
printf("The pt is present...\n");

if(x==0)
 printf("The pt is absent...\n");

        for (i=0; i<2; i++)
```

```
        {
                        printf("Delete which coordinate\n");
                        scanf("%d",&pt2[i]);
        }
        root = delete(root,pt2);
                preOrder(root);
}
```