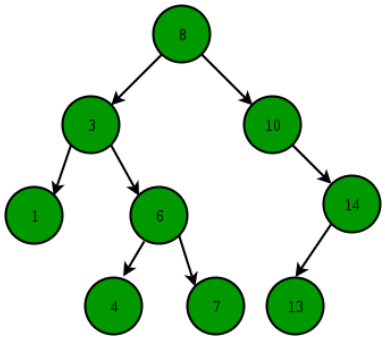
1. Tartiblangan va muvozanatlashgan daraxtlar

Ikkilik qidiruv daraxti tugunga asoslangan ikkilik daraxt ma’lumotlar tuzilishi quyidagi xususiyatlarga ega:

- Tugunning chap pastki qismida faqat tugun kalitidan kichik qiymatlarga ega tugunlar mavjud.
- Tugunning o‘ng pastki daraxti faqat tugun kalitidan kattaroq qiymatlarga ega tugunlarni o‘z ichiga oladi.

Bu shuni anglatadiki, ildizning chap tomonidagi hamma narsa ildiz qiymatidan kamroq va ildizning o‘ng tomonidagi hamma narsa ildiz qiymatidan kattaroqdir. Ushbu ijro tufayli ikkilik qidirish juda oson.

- Ikki nusxadagi tugunlar bo‘lmasligi kerak (lekin turli xil ishlov berish yondashuvlari bilan takroriy qiymatlarga ega bo‘lishi mumkin)



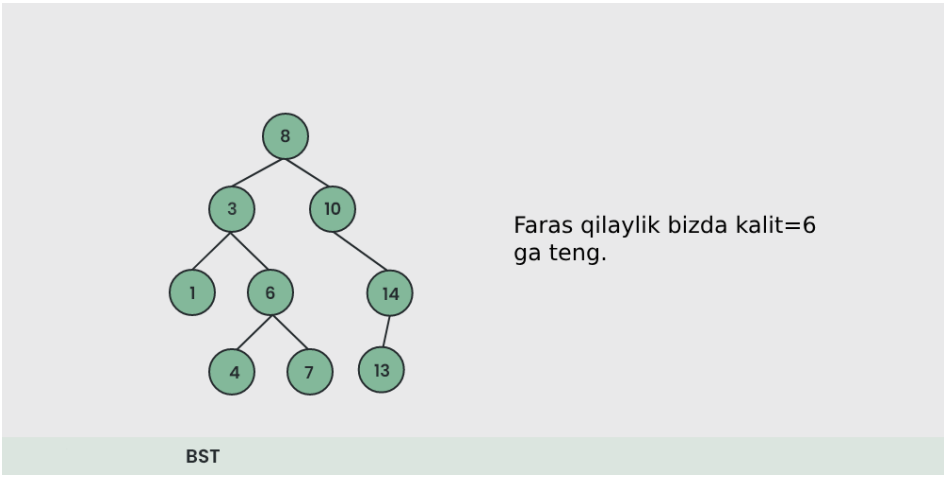
Rasm- 1 Ikkilik qidiruv daraxti tugunga asoslangan ikkilik daraxt ma’lumotlar tuzilishi xususiyatlari.

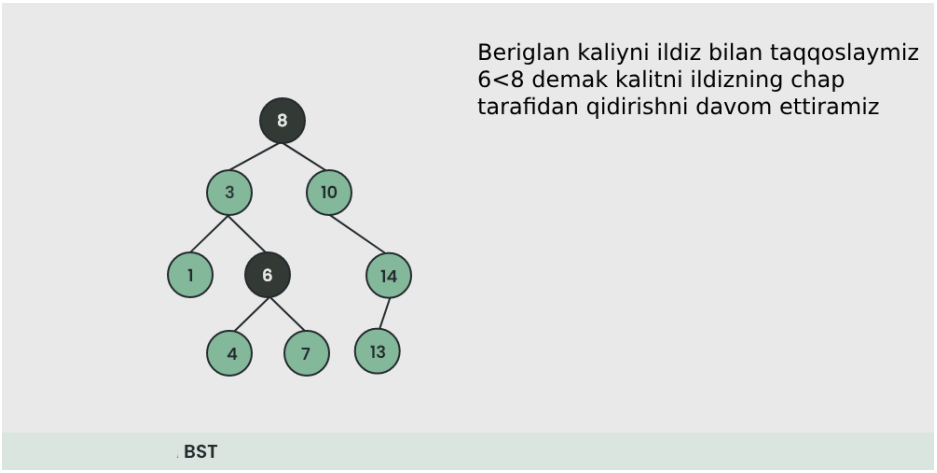
1.1 Ikkilik qidiruv daraxtida qidirish (BST)

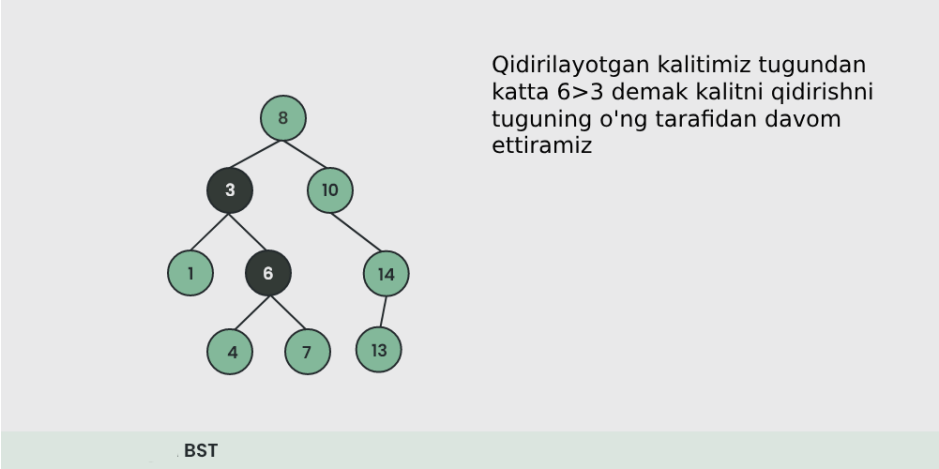
BST-da qiymatni qidirish uchun uni saralangan massiv sifatida ko‘rib chiqamiz. Endi biz BST-da qidiruv operatsiyasini osongina bajarishimiz mumkin.

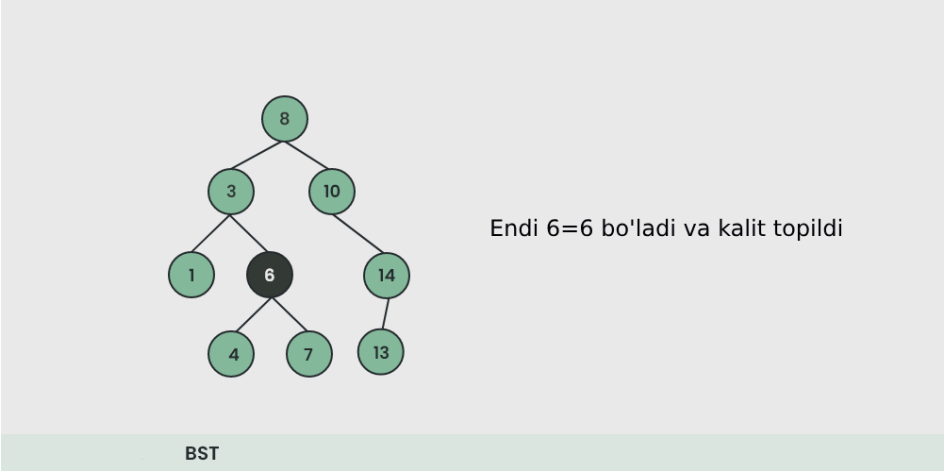
Aytaylik, biz x raqamini qidirmoqchimiz, biz ildizdan boshlaymiz. Keyin:Biz qidiriladigan qiymatni ildiz qiymati bilan taqqoslaymiz. Agar u teng bo‘lsa, biz qidiruvni tugatamiz agar u kichikroq bo‘lsa, biz chap pastki daraxtga o‘tishimiz kerakligini bilamiz, chunki ikkilik qidiruv daraxtida chap pastki daraxtdagi barcha

elementlar kichikroq va o‘ng pastki daraxtdagi barcha elementlar kattaroqdir. Boshqa o‘tish imkoni bo‘lmaguncha yuqoridagi amalni takrorlaymiz.Agar biron-bir iteratsiyada kalit topilsa, True-ni qaytaramiz. Algoritmni yaxshiroq tushunish uchun quyidagi misolni ko‘rib chiqamiz :









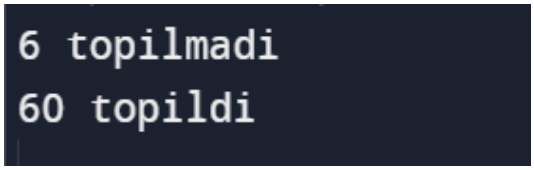
BST qidiruv amalga oshirish uchun C++ dasturlash tilida dastur:

```
#include <iostream>

using namespace std;

struct node {
   int key;
   struct node *left, *right;
};

// Yangi BST tugunini yaratish uchun yordamchi funktsiya
struct node* newNode(int item)
{
   struct node* temp
        = new struct node;
```

```
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}
// Qo‘shish uchun yordamchi funktsiya
// BST-da berilgan kalit bilan yangi tugun
struct node* insert(struct node* node, int key)
{
    // If the tree is empty, return a new node
    // Agar daraxt bo‘sh bo‘lsa yangi tugunni qaytaring
    if (node == NULL)
        return newNode(key);

    // Aks holda takrorlang
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

        return node;
}

struct node* search(struct node* root, int key)
{
        if (root == NULL || root->key == key)
        return root;

    // Agar kalit katta bo‘lsa
    if (root->key < key)
        return search(root->right, key);

    // Agar kalit kichkina bo‘sa
    return search(root->left, key);
}

// Bosh funktsiya
int main()
{
    struct node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
```

```
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // kalit
    int key = 6;

    // BST qidirish
    if (search(root, key) == NULL)
        cout << key << " topilmadi" << endl;
    else
        cout << key << " topildi" << endl;

    key = 60;

    // BST qidirish
    if (search(root, key) == NULL)
        cout << key << " topilmadi" << endl;
    else
        cout << key << " topildi" << endl;
    return 0;
}
```

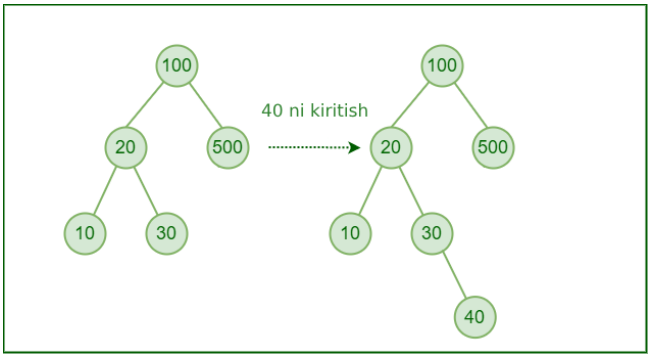
Dastur natijasi:



Vaqt murakkabligi: O(h), bu yerda h daraxt balandligi.

Xotira murakkabligi: O(h), bu erda h daraxt balandligi. Buning sababi shundaki, rekursiya to‘plamini saqlash uchun zarur bo‘lgan maksimal bo‘sh joy baxtli bo‘ladi.

1.2 Ikkilik qidiruv daraxtiga element kiritish (BST)

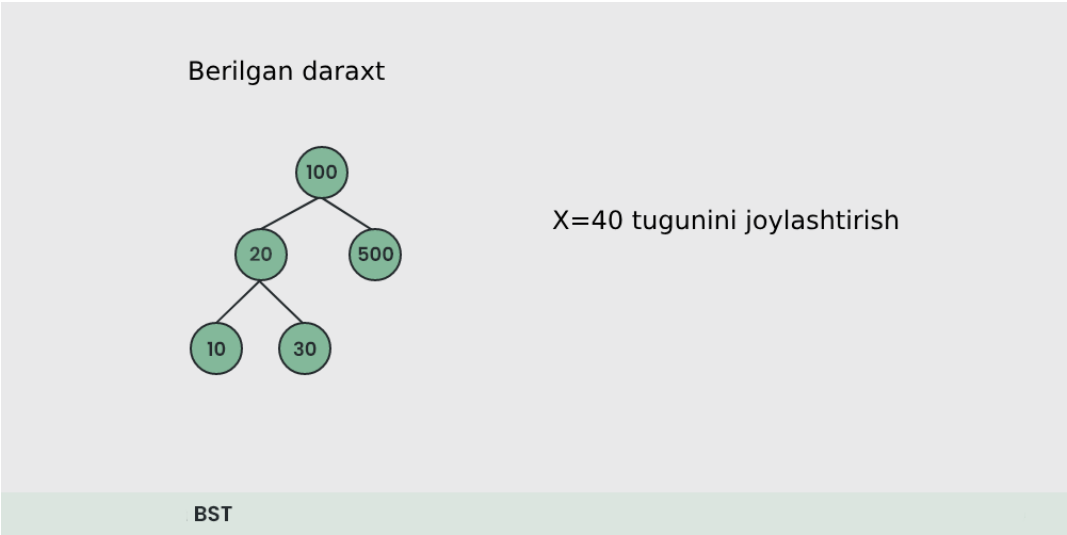


Rasm- 2 Ikkilik daraxtga element kiritish

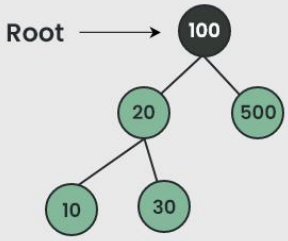
Yangi kalit har doim oxirgi nuqtaga kiritiladi va ikkilik qidiruv daraxti xususiyatini saqlab qoladi. Biz oxirgi tugunga yetguncha kalitni ildizdan qidirishni boshlaymiz. Oxirgi tugun topilgandan so‘ng, yangi tugun oxirgi tugunga nisbatan voris sifatida qo‘shiladi. Quyidagi qadamlar tugunni ikkilik qidiruv daraxtiga kiritishga harakat qilganda amalga oshiriladi:

Kiritiladigan qiymatni (aytaylik, X) biz joylashgan joriy tugun qiymatiga moslang:

Agar X qiymatidan dan kichikroq bo‘lsa, chap pastki daraxtga o‘tiladi.Aks holda, o‘ng pastki daraxtga o‘tiladi.Yakuniy tugunga erishilgandan so‘ng, x va oxirgi tugun qiymati o‘rtasidagi nisbatga qarab uning o‘ng yoki chap tomoniga x ni qo‘yamiz.

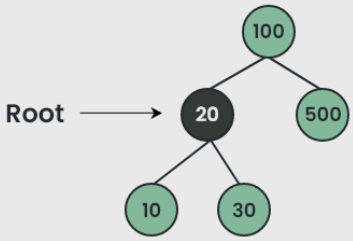


Qadam-1:ildiz bilan tugunni taqqoslaymiz



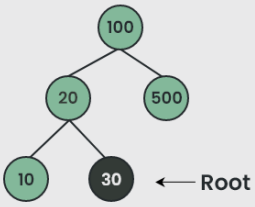
100>40 bo'glanligi sababli
ko'rsatkich chapka o'tadi

Qadam-2: Chap voirs bilan kalit taqqoslanadi

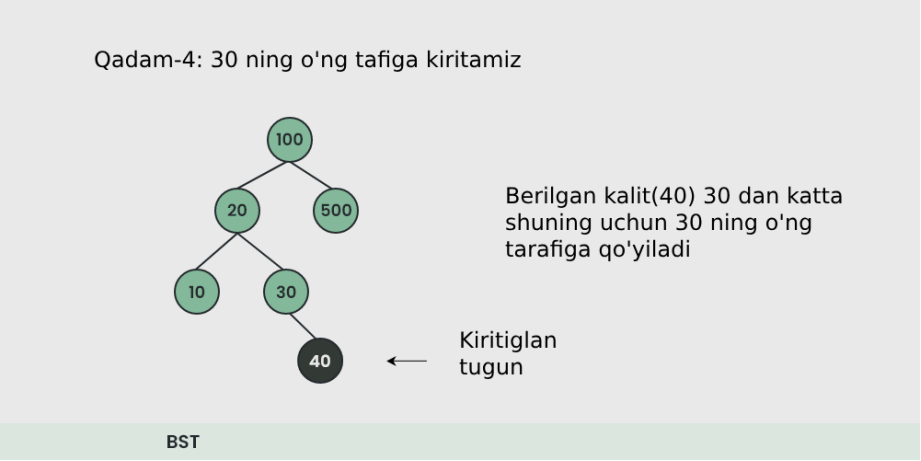


20<40 ko'rsatkichni o'nga otkazamiz(30)

Qadam-3: kalitni 20 ning o'ng tarafidagi
vorisi bilan taqqoslaymiz.



Yana 40 30 dan katta
ko'rsatkichni 30 ning
o'ng tarafiga o'tqizamiz.



C++ dasturlash tili rekursiya yordamida ikkilik qidiruv daraxtiga tugun kiritish:

```
#include <bits/stdc++.h>
using namespace std;

class BST {
   int data;
   BST *left, *right;

public:
   // Konstruktor
   BST();

   BST(int);

   // Funktsiyani kiritish
   BST* Insert(BST*, int);

        void Inorder(BST*);
};

BST::BST()
   : data(0)
   , left(NULL)
   , right(NULL)
{
}

BST::BST(int value)
```

```
{
   data = value;
   left = right = NULL;
}

BST* BST::Insert(BST* root, int value)
{
   if (!root) {

                  return new BST(value);
   }

   // Ma'lumotlarni kiritish
   if (value > root->data) {

                  root->right = Insert(root->right, value);
   }
   else if (value < root->data) {

   root->left = Insert(root->left, value);
   }

        return root;
}
void BST::Inorder(BST* root)
{
   if (!root) {
        return;
   }
   Inorder(root->left);
   cout << root->data << " ";
   Inorder(root->right);
}

// Asosiy funktsiya
int main()
{
   BST b, *root = NULL;
   root = b.Insert(root, 50);
   b.Insert(root, 30);
   b.Insert(root, 20);
```

```
    b.Insert(root, 40);
    b.Insert(root, 70);
    b.Insert(root, 60);
    b.Insert(root, 80);

    b.Inorder(root);
    return 0;
}
```

Dastur natijasi:

```
/tmp/Y1GcFSfqbR.o
20 30 40 50 60 70 80
```

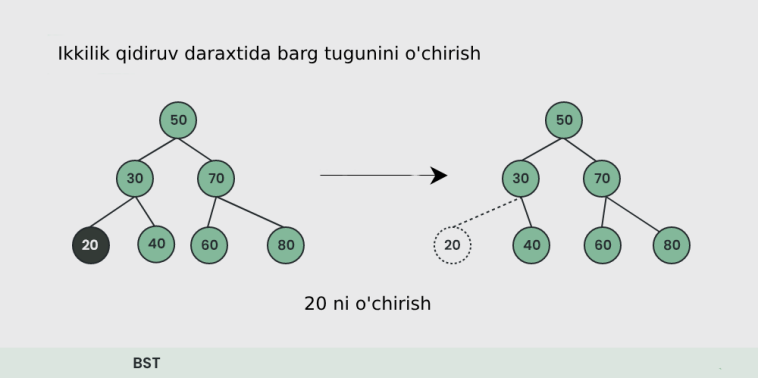
Vaqt murakkabligi: O(n) ga teng, chunki har bir tugun bir marta tashrif buyuradi.

Xotira murakkabligi: O(n) ga teng, chunki biz rekursiya uchun tugunlarni saqlash uchun stackdan foydalanamiz.

1.3 Ikkilik qidiruv daraxtida elementni o‘chirish (BST)

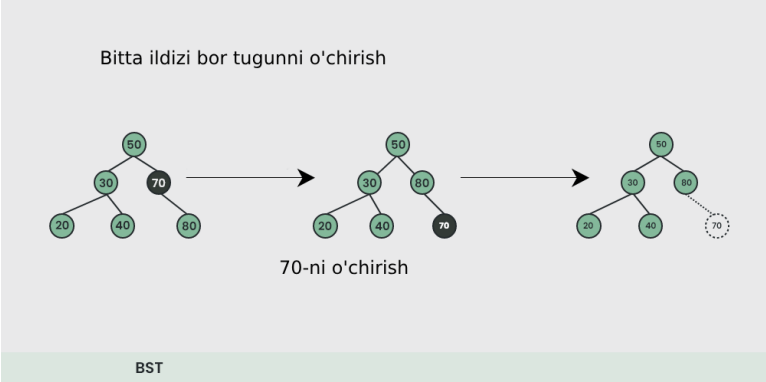
Ikkilik qidiruv daraxtida o‘chirishni 3 xil turda ko‘rishimiz mumkin.

1. BST-dagi barg tugunini o‘chirish:



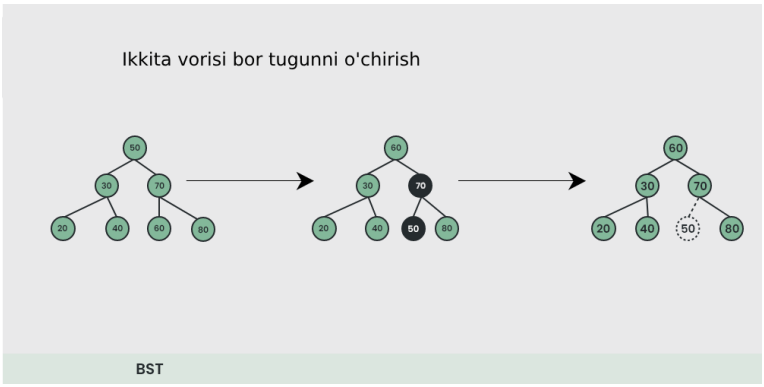
Rasm- 3 ikkilik qidiruv daraxtida bar tugunini o‘chirish

2. BST bitta vorisi bor tugunni o‘chirish: BSTda bitta vorisi bor tugunini o‘chirish oddiy. Tugunni voris bilan o‘rnini almashtirish va tugunni o‘chirish.



3.Ikkita vorisi bor tugunni o‘chirish.

Vorislar bilan tugunni olib tashlash unchalik oson emas. Bu erda biz tugunni shunday olib tashlashimiz kerakki, hosil bo‘lgan daraxt quyidagi xususiyatlarni saqlab qosin.



Ikkillik qidiruv darxtidan tugunni o‘chirishing C++ das turlash tilida implementatsiyasi.

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
   int key;
   struct Node *left, *right;
};

Node* newNode(int item)
{
   Node* temp = new Node;
   temp->key = item;
```

```
    temp->left = temp->right = NULL;
    return temp;
}
void inorder(Node* root)
{
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}
Node* insert(Node* node, int key)
{
        if (node == NULL)
        return newNode(key);

        if (key < node->key)
        node->left = insert(node->left, key);
    else
        node->right = insert(node->right, key);
    return node;
}
Node* deleteNode(Node* root, int k)
{

    if (root == NULL)

    if (root->key > k) {
        root->left = deleteNode(root->left, k);
        return root;
    }
    else if (root->key < k) {
        root->right = deleteNode(root->right, k);
        return root;
    }

    if (root->left == NULL) {
        Node* temp = root->right;
        delete root;
        return temp;
    }
```

```
    else if (root->right == NULL) {
        Node* temp = root->left;
        delete root;
        return temp;
    }

    else {

        Node* succParent = root;

        // Find successor
        Node* succ = root->right;
        while (succ->left != NULL) {
            succParent = succ;
            succ = succ->left;
        }
            succParent->left = succ->right;
        else
            succParent->right = succ->right;

        root->key = succ->key;
        delete succ;
        return root;
    }
}
int main()
{
    Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);

    printf("Berilgan BST: ");
    inorder(root);

    printf("\n\nBST daraxtida barg (20) ni o‘chirish: \n");
    root = deleteNode(root, 20);
    printf("BST daraxtidan bargni o‘chirgandan keyin: \n");
```

```
    inorder(root);

    printf("\n\nBita vorisli tugunni o‘chirish(70): \n");
    root = deleteNode(root, 70);
    printf("O‘chirlgandan keyingi holat: \n");
    inorder(root);

    printf("\n\nIkkita avlodli tugunni o‘chirish: \n");
    root = deleteNode(root, 50);
    printf("Tugunni o‘chirishtan keyin: \n");
    inorder(root);

    return 0;
}
```

Vaqt murakkabligi: O(h), bu yerda h daraxt balandligi.

Xotira murakkabligi: O(n).

Mavzu yuzasidan savollar:

1. Daraxt ma’lumotlar strukturasiga ta’rif bering
2. Daraxtning eng asosiy tushunchalariga to‘xtalib o‘ting.
3. Pryufer kodini hosil qilish va qo‘lllanishi haqida gapiring
4. Pryufer kodi asosida daraxtni tiklash qanday amalga oshiriladi?
5. Daraxt ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?