1. AVL daraxti ma’lumotlari tuzilmasi

O‘zini muvozanatlashtiradigan ikkilik qidiruv daraxti (BST) AVL daraxti deyiladi. Bu erda har qanday tugun uchun chap va o‘ng pastki daraxt balandliklari orasidagi farq muvozanat omili birdan ortiq bo‘lishi mumkin emas.

AVL daraxti uning ixtirochilari Georgiy Adelson-Velskiy va Evgenii Landis sharafiga nomlangan, ular birinchi 1962 yilda “axborotni tashkil etish algoritmi” maqolasida nashr etishgan.

Balans Omili (Balance factor (BF))

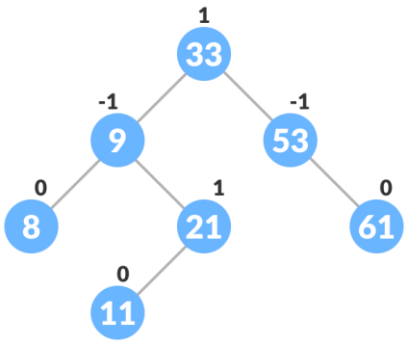
Avl daraxtidagi tugunning muvozanat koeffitsienti-bu chap pastki daraxt balandligi va ushbu tugunning o‘ng pastki daraxti o‘rtasidagi farq.Balans omilining formulasi quydagicha :

$$BF=Lh-Rh$$

- BF->Balans omili(balance factor)
- Lh->Tugunning chap vorisi balandligi
- Rh->Tugunning o‘ng vorisi balandligi

Agar BF={-1;0;1} bo‘lsa daraxt muvozanatlashgan bo‘ladi.

Muvozanatli avl daraxtiga misol:



Rasm- 1 Balans omili

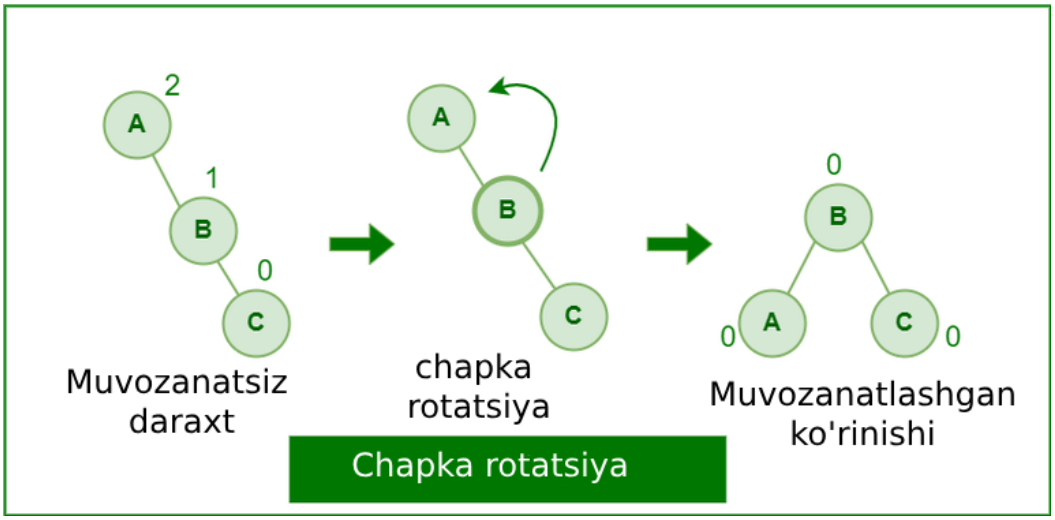
Masalan ildiz daraxti tugining chap tarafidagi balandlik 3 ka o‘ng tarafi balandligi 2ga teng. Shunda BF(33)=3-1=1.

1.1 AVL daraxtidagi bajariladigan amallar

AVL daraxtida rotatsiyalarni amalga ochirishining to‘rt turi mavjud.

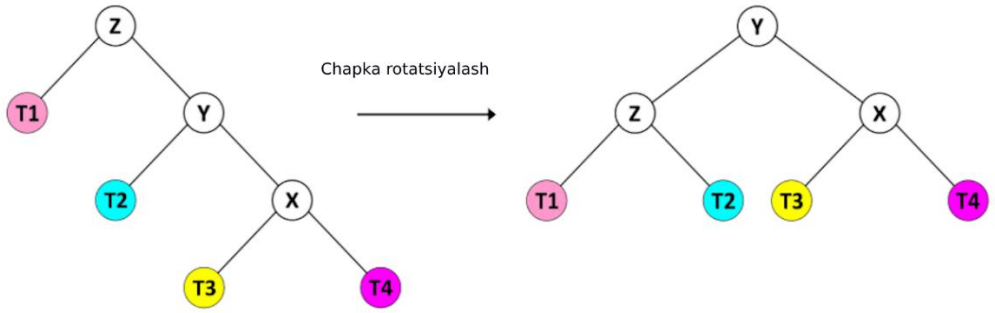
Chapka rotatsiya

Chapga rotatsiyada tugunlarning o‘ngdagi joylashuvi chap tugundagi tartibga aylantiriladi. Agar tugun o‘ng pastki daraxtning o‘ng pastki qismiga qo‘shilsa va daraxt muvozanatdan chiqsa, biz bitta chap rotatsiyasini bajaramiz.



Rasm- 2 chapka rotatsiya

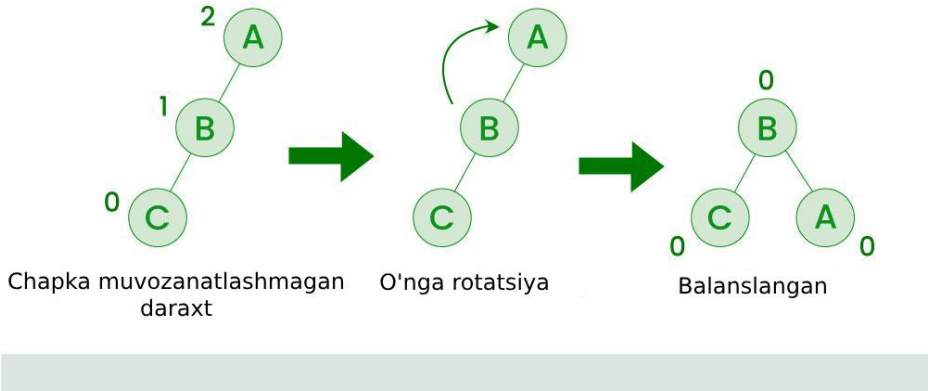
Agar rotatsiya qilinayotgan tugunning vorisi bo‘lsa u holda:



Rasm- 3 chapka rotatsiya tugunda voris bo‘lsa

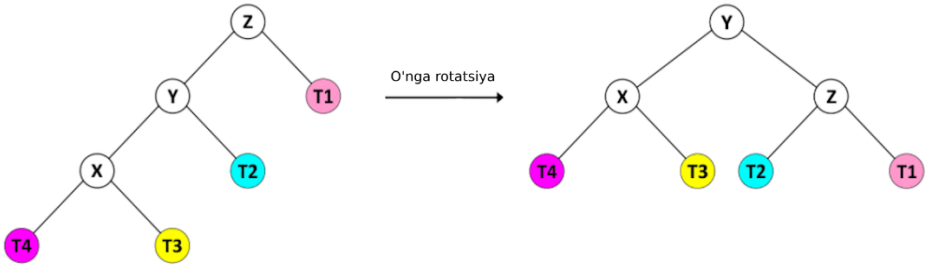
O‘nga rotatsiya:

Agar chap pastki daraxtning chap pastki qismiga tugun qo‘shilsa, AVL daraxti muvozanatdan chiqib ketishi mumkin, biz bitta o‘nga rotatsiyasini bajaramiz.



Rasm- 4 O‘nga rotatsiya

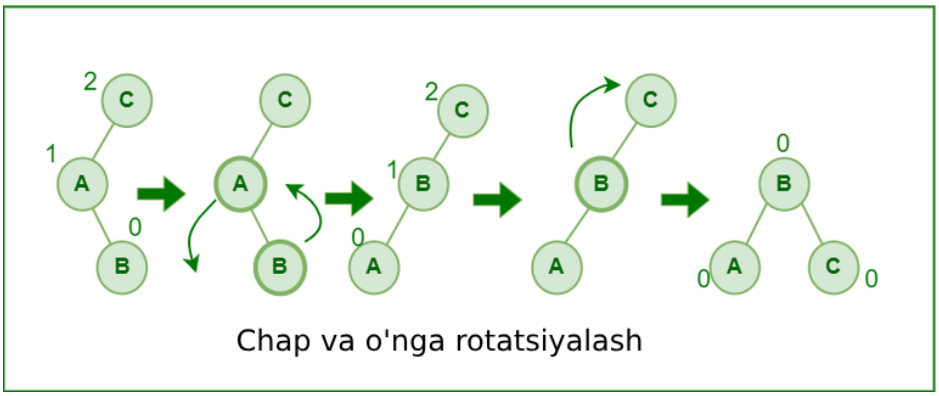
Agar rotatsiya qilinayotgan tugunning vorisi bo‘lsa u holda:



Rasm- 5 O‘nga rotatsiya agar tugunda voris bo‘lsa

Chap - o‘ngga rotatsiya:

Chap-o‘ng rotatsiyasi-bu o‘ng aylanish bajarilgandan keyin birinchi chap aylanishi sodir bo‘ladigan kombinatsiya.

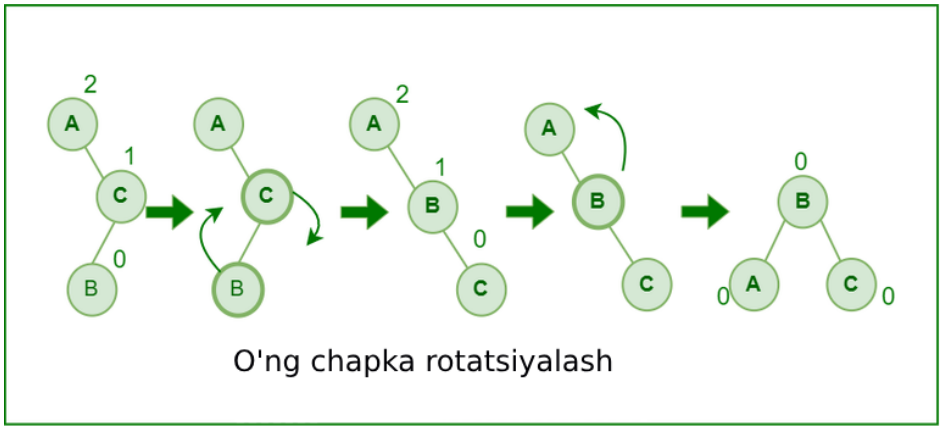


Rasm- 6 Chap va o‘nga rotatsiya

Agar rotatsiya qilinayotgan tugunning vorisi bo‘lsa u holda:

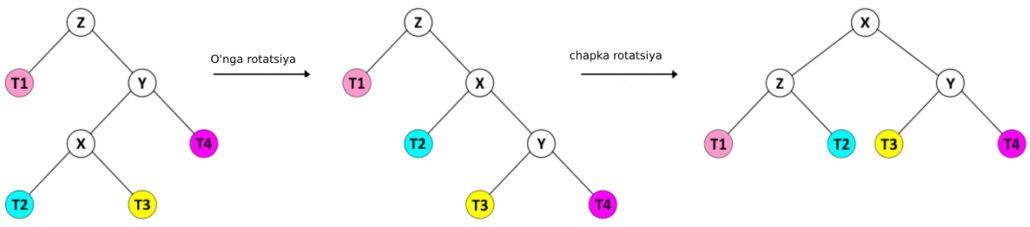
O‘ng-Chapka rotatsiyalash:

O‘ng-chap aylanish-bu chap aylanish bajarilgandan keyin birinchi o‘ng aylanishi sodir bo‘ladigan kombinatsiya.



Rasm- 7 O‘nga va chapka rotatsiya

Agar rotatsiya qilinayotgan tugunning vorisi bo‘lsa u holda:

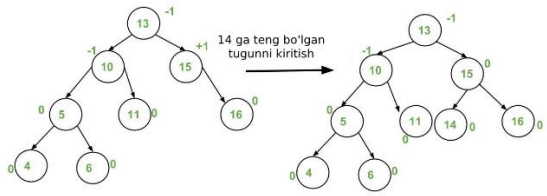
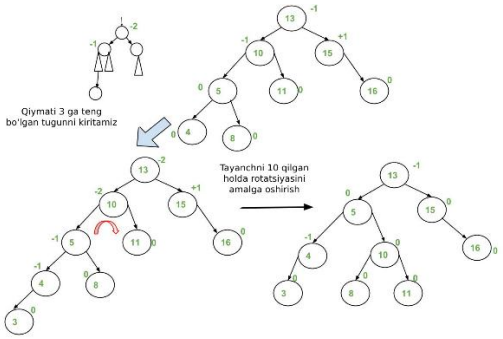
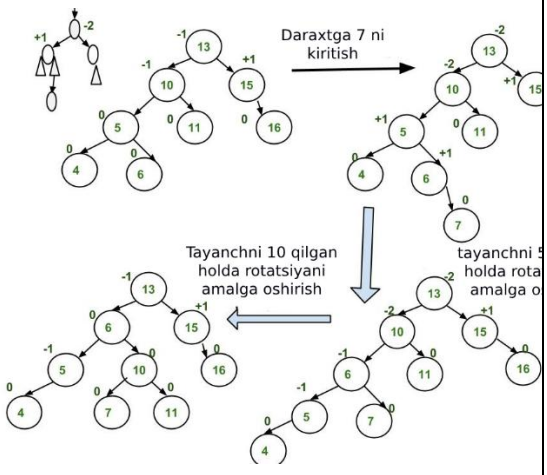


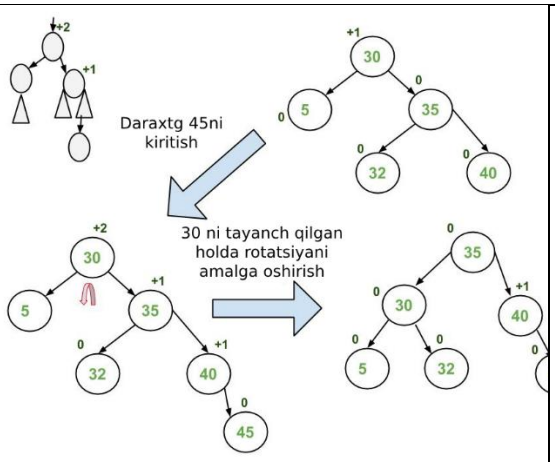
Rasm- 8 Agar rotatsiya qilinayotgan tugunning vorisi bo‘lsa

1.2 AVL daraxtiga yangi tugun qo‘shish

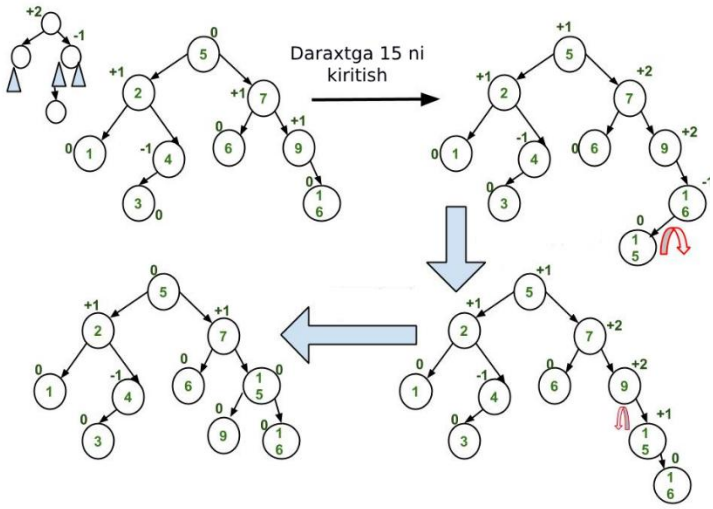
AVL daraxtining balandligi har doim O(log(n)) ga teng. Bu erda n-daraxtdagi tugunlar soni. Quyida BST xususiyatini buzmasdan BSTni muvozanatlash uchun bajarilishi mumkin bo‘lgan ikkita asosiy operatsiya keltirilgan (kalitlar(chapda) < kalit(ildiz) < kalitlar(o‘ngda)).

Eng ko‘p BST algoritmda bajariladigan operatsialar (qidirish, kiritish o‘chirish...) O(h) vaqtini oladi. Bu yerda h daraxtning balandligi.Agar biz daraxt balandligini O(log(n)) ga tushira olsak (n-daraxt balandligi) bajariladigan operatsialar sezilarli darajada tezlashadi.Pastda berilgan misollarda AVL daraxtga yangi tugun kiritishning bir nechta holatlarini ko‘rishimiz mumkin.

<table>
<tr>
<td>1).
 
</td>
<td>2)
 
</td>
</tr>
<tr>
<td>3)</td>
<td>4)
 
</td>
</tr>
</table>



5.



Berilgan amallarning bajarish ketma ketligi:

Joriy tugun yangi kiritilgan tugunning ajdodlaridan biri bo‘lishi kerak. Joriy tugunning balandligini yangilang.

Joriy tugunning muvozanat koeffitsientini (chap pastki daraxt balandligi – o‘ng pastki daraxt balandligi) oling.

Agar muvozanat koeffitsienti 1 dan katta bo‘lsa, u holda joriy tugun muvozanatsiz va biz chap chap holatda yoki chap o‘ng holatda bo‘lamiz. Chap

chap holatda yoki yo‘qligini tekshirish uchun yangi kiritilgan kalitni chap voris ildizidagi kalit bilan solishtiring.

Agar muvozanat koeffitsienti -1 dan kam bo‘lsa, u holda joriy tugun muvozanatsiz va biz o‘ng o‘ng holatda yoki o‘ng-chap holatda bo‘lamiz. Bu to‘g’ri o‘ng holat yoki yo‘qligini tekshirish uchun yangi kiritilgan kalitni o‘ng pastki ildizdagi kalit bilan Solishtiring.

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

// AVl daraxt tuguni
class Node
{
   public:
   int key;
   Node *left;
   Node *right;
   int height;
};
//daraxt balanligini
//hisoblovchi funktsiya
int height(Node *N)
{
   if (N == NULL)
        return 0;
   return N->height;
}
//maximumni topish funkysiyasi
int max(int a, int b)
{
   return (a > b)? a : b;
}
//yordamchi funktsiya
Node* newNode(int key)
{
   Node* node = new Node();
   node->key = key;
   node->left = NULL;
   node->right = NULL;
```

```
    node->height = 1;
    return(node);
}

// O‘nga rotatsiya
Node *rightRotate(Node *y)
{
    Node *x = y->left;
    Node *T2 = x->right;

    // Rotatsiyani amalga oshirish
    x->right = y;
    y->left = T2;

    // Balanligini yangilash
    y->height = max(height(y->left),
                    height(y->right)) + 1;
    x->height = max(height(x->left),
                    height(x->right)) + 1;

    // Yangi ildiz qaytarsish
    return x;
}
//chapka rotatsiyalash
Node *leftRotate(Node *x)
{
    Node *y = x->right;
    Node *T2 = y->left;

    // Rotatsiyani amalga oshirish
    y->left = x;
    x->right = T2;

    // Balandligini yangilash
    x->height = max(height(x->left),
                    height(x->right)) + 1;
    y->height = max(height(y->left),
                    height(y->right)) + 1;

    // Ildizni qaytarish
    return y;
```

```
}

// Balance faktorni hisoblash
int getBalance(Node *N)
{
   if (N == NULL)
        return 0;
   return height(N->left) - height(N->right);
}
//Kalitni kiritish uchun funktsiya
Node* insert(Node* node, int key)
{
   // BST tugunini kiritish
   if (node == NULL)
        return(newNode(key));

   if (key < node->key)
        node->left = insert(node->left, key);
   else if (key > node->key)
        node->right = insert(node->right, key);
   else // Bir biriga teng tugunlar taqiqlanadi
        return node;

   /* 2. Update height of this ancestor node */
   node->height = 1 + max(height(node->left),
                          height(node->right));

   int balance = getBalance(node);


   // Chap chap
   if (balance > 1 && key < node->left->key)
        return rightRotate(node);

   // O‘ng o‘ng
   if (balance < -1 && key > node->right->key)
        return leftRotate(node);

   // Chap o‘ng
   if (balance > 1 && key > node->left->key)
   {
```

```
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    // O‘ng chap
    if (balance < -1 && key < node->right->key)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

// Daraxtni qayta tartiblash
//funktsiyasi
void preOrder(Node *root)
{
    if(root != NULL)
    {
        cout << root->key << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}

// Bosh funktsiya
int main()
{
    Node *root = NULL;

    /* Daraxt tugunlarini joylashtirish */
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);

    /* The constructed AVL Tree would be
```

```
                / \
                20 40
                / \ \
          10 25 50
    */
    cout << "Avl daraxti:  \n";
    preOrder(root);

    return 0;
}
```

Dastur natijasi:

Vaqt murakkabligi: O(log (n)).

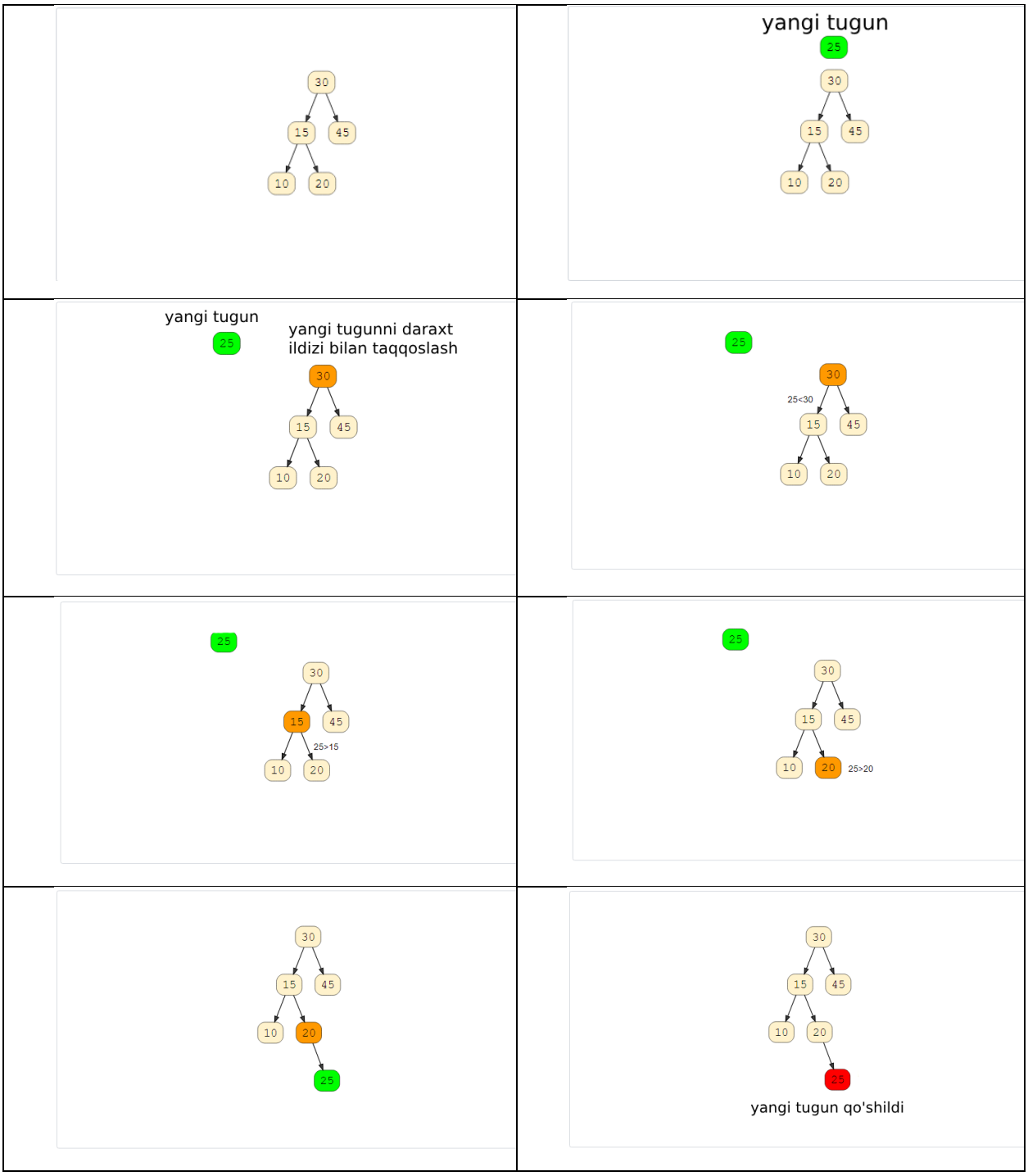
Xotira murakkabligi: O(1)

1.3 AVL daraxtidan tugunni o‘chirish

AVL daraxtiga yangi tugunni samarali kiritish 2 ta asosiy bosqichni o‘z ichiga olgan rekursiv yondashuvni talab qiladi:

Tugunni kiritish: biz tugunni daraxtga ikkilik qidiruv daraxtiga kiritilgandek kiritamiz—rekursiv ravishda. Ushbu yangi tugun barg tuguniga aylanadi.

AVL-ga tugunni kiritish uchun avval qiymatini taqqoslaymiz x (tugun kiritilmoqda) ildiz tugunining qiymati bilan. Agar bu qiymat ildiz tugunining qiymatidan kam bo‘lsa, biz chap bolaga o‘tamiz va uni keyingi rekursiv chaqirish uchun ildiz tuguniga aylantiramiz. Xuddi shunday, agar bu qiymat ildiz tugunining qiymatidan katta bo‘lsa. Biz o‘ng vorisga boramiz va uni keyingi rekursiv chaqirish uchun ildiz tuguniga aylantiramiz. Biz barg tuguniga yetguncha bu rekursiv qo‘ng’iroqlarni davom ettiramiz. Quyidagi misol buni ko‘rsatadi (rekursiv chaqiruvlar paytida joriy ildiz tuguni to‘q sariq rangga bo‘yalgan):

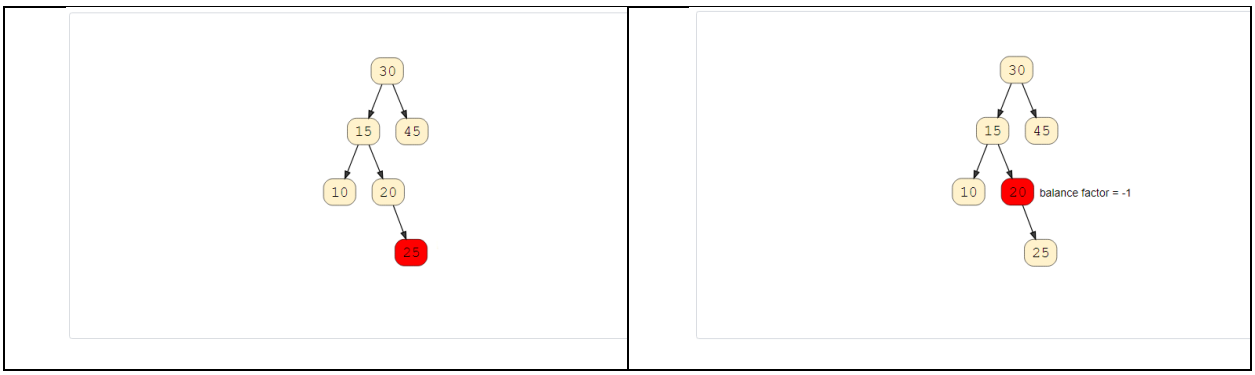


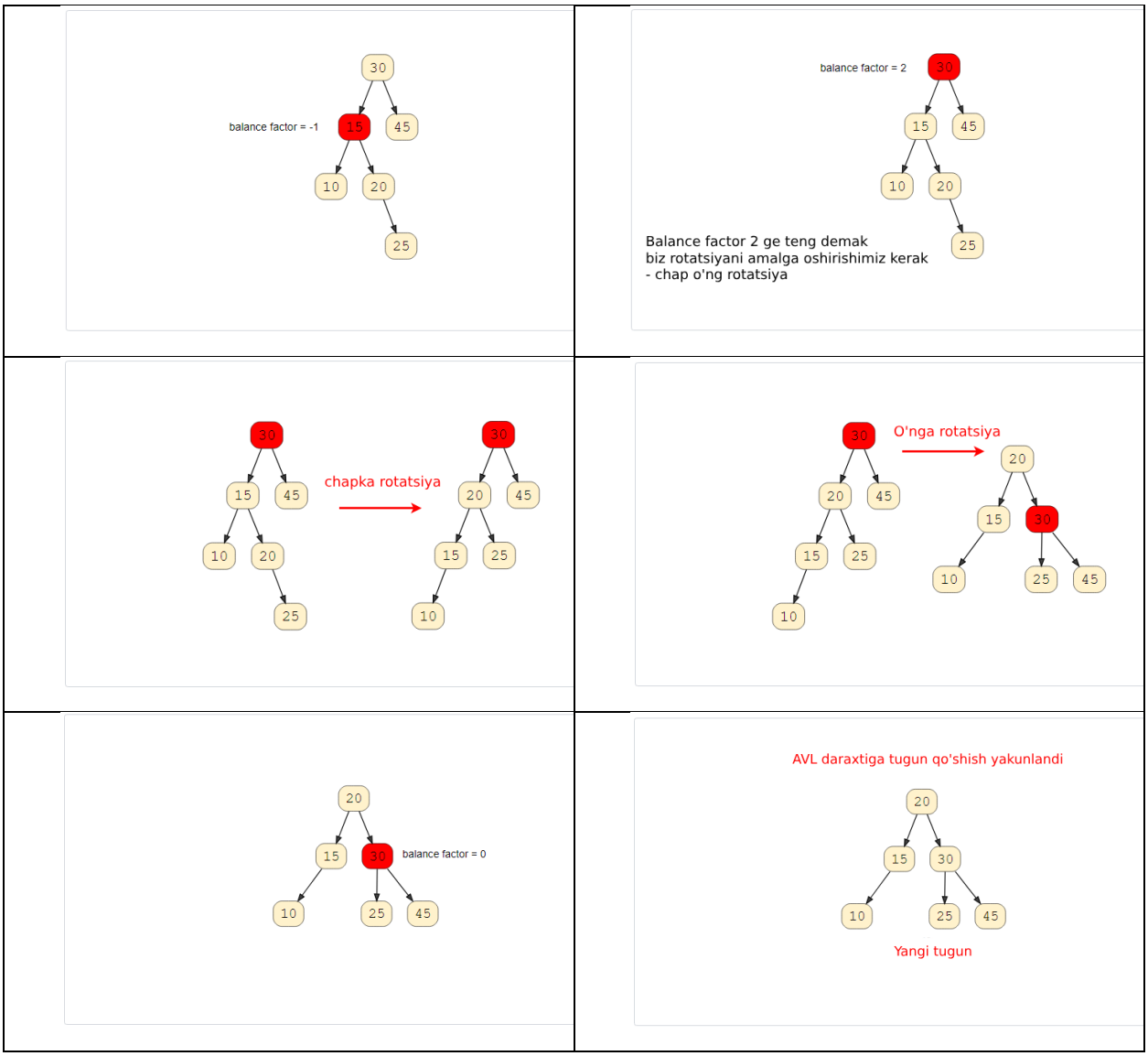
<u>Tugun kiritilgandan so‘ng daraxtni muvozanatlash</u>: bu orqaga qaytish boshlangandan so‘ng sodir bo‘ladi. Yangi kiritilgan tugundan (hozirda barg tuguni) boshlab, biz birinchi muvozanatsiz tugun ([-1, 1] diapazondan tashqarida joylashgan muvozanat koeffitsienti bilan) topilmaguncha daraxt bo‘ylab harakatlanamiz.

Keyinchalik, biz ushbu muvozanatsiz tugunning pastki qismida uni muvozanatlash uchun kerakli rotatsiyalarni bajaramiz. Bu daraxtning ildiz tuguniga etib borgunimizcha va rekursiyani orqaga qaytarish tugaguniga qadar takrorlanadi.

Agar muvozanat koeffitsienti 2 bo‘lsa,demak, bu muvozanatsiz tugunning chap pastki daraxti o‘ngdan og’irroq. Shunday qilib, chap-chap nomutanosiblik yoki chap-o‘ng nomutanosiblik mavjud. Endi, agar muvozanatsiz tugunning chap vorisining muvozanat koeffitsienti 1 dan katta yoki unga teng bo‘lsa (yoki muqobil ravishda, agar x qiymati muvozanatsiz tugunning chap vorisi qiymatidan kam bo‘lsa), bu uni chap-chap nomutanosiblikgacha olib keladi va muvozanatsiz tugunda o‘ng rotatsiyasini qo‘llash kerak, aks holda nomutanosiblik chap-o‘ng rotatsiyasini qo‘llash kerak.

Xuddi shunday, agar muvozanat koeffitsienti -2 bo‘lsa, bu muvozanatsiz tugunning o‘ng pastki daraxti chapdan og’irroq ekanligini anglatadi. Shunday qilib, o‘ng-o‘ng nomutanosiblik yoki o‘ng-chap nomutanosiblik mavjud. Endi, agar muvozanatsiz tugunning o‘ng vorisining muvozanat koeffitsienti 0 dan kam yoki unga teng bo‘lsa (yoki muqobil ravishda, agar x qiymati muvozanatsiz tugunning o‘ng bolasi qiymatidan katta bo‘lsa), bu uni o‘ng-o‘ng nomutanosiblikgacha olib keladi va muvozanatsiz tugunda chap rotatsiyani qo‘llash kerak. Aks holda, nomutanosiblik o‘ng-chap rotatsiyasini qo‘llash kerak. Tepadagi berilgan misolni davom ettiramiz.





Kiritish uchun: Vaqt murakkabligi: O(log (n)), Xotira murakkabligi: O(1)

```
#include<iostream>
using namespace std;
// Bir AVL daraxti tuguni
class Node
{
   public:
   int key;
   Node *left;
   Node *right;
   int height;
};
```

```
// Olish uchun yordamchi funktsiya
// daraxtning balandligi
int height(Node *N)
{
   if (N == NULL)
        return 0;
   return N->height;
}
// ikki butun sondanMaksimal foyda
//olish uchun yordamchi funktsiya
int max(int a, int b)
{
   return (a > b)? a : b;
}

Node* newNode(int key)
{
   Node* node = new Node();
   node->key = key;
   node->left = NULL;
   node->right = NULL;
   node->height = 1;

   return(node);
}
Node *rightRotate(Node *y)
{
   Node *x = y->left;
   Node *T2 = x->right;

   // Rotatsiyani amalga oshirish
   x->right = y;
   y->left = T2;

   // balandligini o‘zgartirish
   y->height = max(height(y->left),
                   height(y->right)) + 1;
   x->height = max(height(x->left),
                   height(x->right)) + 1;

   // yangi ildiz qaytarish
```

```
    return x;
}

Node *leftRotate(Node *x)
{
    Node *y = x->right;
    Node *T2 = y->left;

    // Rotatsiyani amalga oshirish
    y->left = x;
    x->right = T2;

    // balandligini o‘zgartirish
    x->height = max(height(x->left),
                    height(x->right)) + 1;
    y->height = max(height(y->left),
                    height(y->right)) + 1;

    // yangi ildiz qaytarish
    return y;
}

// Balance faktorni tekshirish
int getBalance(Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}
Node* insert(Node* node, int key)
{
    /* 1. BST kiritishni amalaga oshirish */
    if (node == NULL)
        return(newNode(key));

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else // Equal keys are not allowed in BST
        return node;
```

```
    /* 2. Daraxt balandligini o‘zgartirish*/
    node->height = 1 + max(height(node->left),
                           height(node->right));

    /* 3. Tugun qo‘shilgandan so‘ng
    balance ni tekshirish*/
    int balance = getBalance(node);

    // muvozanat buzilsagandagi 4 holat
    // Chap chap holat
    if (balance > 1 && key < node->left->key)
         return rightRotate(node);
    // O‘ng o‘ng holat
    if (balance < -1 && key > node->right->key)
         return leftRotate(node);
    // Chap o‘ng holat
    if (balance > 1 && key > node->left->key)
    {
         node->left = leftRotate(node->left);
         return rightRotate(node);
    }
    // o‘ng chap holat
    if (balance < -1 && key < node->right->key)
    {
         node->right = rightRotate(node->right);
         return leftRotate(node);
    }
    return node;
}
void preOrder(Node *root)
{
    if(root != NULL)
    {
         cout << root->key << " ";
         preOrder(root->left);
         preOrder(root->right);
    }
}

// Driver Code
```

```
int main()
{
   Node *root = NULL;

   /* Daraxt tugunlarini qo‘shish*/
   root = insert(root, 10);
   root = insert(root, 20);
   root = insert(root, 30);
   root = insert(root, 40);
   root = insert(root, 50);
   root = insert(root, 25);

   /* Daraxt ko‘rinishi
                 30
           / \
           20 40
           / \ \
             10 25 50
   */
   cout << "AVL daraxti \n";
   preOrder(root);

   return 0;
}
```

Dastur natijasi:

AVL daraxtining qo‘llanilishi:

- U ma’lumotlar bazasidagi katta matnlarni indekslash va shu bilan samarali qidirish uchun ishlatiladi.
- Xotiradagi to‘plamlarning barcha turlari, shu jumladan to‘plamlar va lug’atlar uchun AVL daraxtlari ishlatiladi.
- Ma’lumotlar bazasi ilovalari, bu erda qo‘shimchalar va o‘chirishlar kamroq uchraydi, lekin tez-tez ma’lumotlarni qidirish uchun zarur.

AVL daraxtining afzalliklari:

- AVL daraxt tugunlari o‘z-o‘zlarini muvozanatlashi mumkin.
- Bu B daraxtlarga qaraganda tezroq qidirishni ta’minlaydi
- Ikkilik daraxt kabi boshqa daraxtlarga nisbatan yaxshiroq qidirish vaqtining murakkabligi.
- Balandlik log(n) dan oshmasligi kerak, bu erda N-daraxtdagi tugunlarning umumiy soni.

AVL daraxtining kamchiliklari:

- Buni amalda bajarish qiyinchilikni tug’diradi.
- Ba’zi operatsiyalar uchun yuqori doimiy omillarga ega.
- B daraxtlarga nisbatan kamroq ishlatiladi.
- AVL daraxtlari juda muvozanat tufayli kiritish va olib tashlash operatsiyalarini bajarish murakkab, chunki ko‘proq rotatsiyalar amalga oshiriladi.

Mavzu yuzasidan savollar:

- 1. AVL daraxti nima?
- 2. Uchlarni muvozanatlash deganda nimani tushunasiz.
- 3. Daraxt ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?
- 4. AVL daraxtida bajariladigan operatsiyalar
- 5. AVL daraxtida tugunlarni kiritish va o‘chirishni tushintiring.